intel® ADVANCE INFORMATION

iAPX 286/10 HIGH PERFORMANCE MICROPROCESSOR WITH MEMORY MANAGEMENT AND PROTECTION

- **High Performance 8 and 10 MHz Processor (Up to six times iAPX 86)**
- **Large Address Space:**
  - **—16 Megabytes Physical**
  - **—1 Gigabyte Virtual per Task**
- **Integrated Memory Management, Four-Level Memory Protection and Support for Virtual Memory and Operating Systems**
- **Two iAPX 86 Upward Compatible Operating Modes:**
  - **—iAPX 86 Real Address Mode**
  - **—Protected Virtual Address Mode**
- **Optional Processor Extension:**
  - **—iAPX 286/20 High Performance 80-bit Numeric Data Processor**
- **Complete System Development Support:**
  - **—Development Software: Assembler, PL/M, Pascal, FORTRAN, and System Utilities**
  - **—In-Circuit-Emulator (ICE™-286)**
- **High Bandwidth Bus Interface (8 or 10 Megabyte/Sec)**

The iAPX 286/10 (80286 part number) is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. The 80286 has built-in memory protection that supports operating system and task isolation as well as program and data privacy within tasks. A 10 MHz iAPX 286/10 provides up to six times greater throughput than the standard 5 MHz iAPX 86/10. The 80286 includes memory management capabilities that map up to 2^{30} bytes (one gigabyte) of virtual address space per task into 2^{24} bytes (16 megabytes) of physical memory.

The iAPX 286 is upward compatible with iAPX 86 and 88 software. Using iAPX 86 real address mode, the 80286 is object code compatible with existing iAPX 86, 88 software. In protected virtual address mode, the 80286 is source code compatible with iAPX 86, 88 software and may require upgrading to use virtual addresses supported by the 80286's integrated memory management and protection mechanism. Both modes operate at full 80286 performance and execute a superset of the iAPX 86 and 88's instructions.

The 80286 provides special operations to support the efficient implementation and execution of operating systems. For example, one instruction can end execution of one task, save its state, switch to a new task, load its state, and start execution of the new task. The 80286 also supports virtual memory systems by providing a segment-not-present exception and restartable instructions.

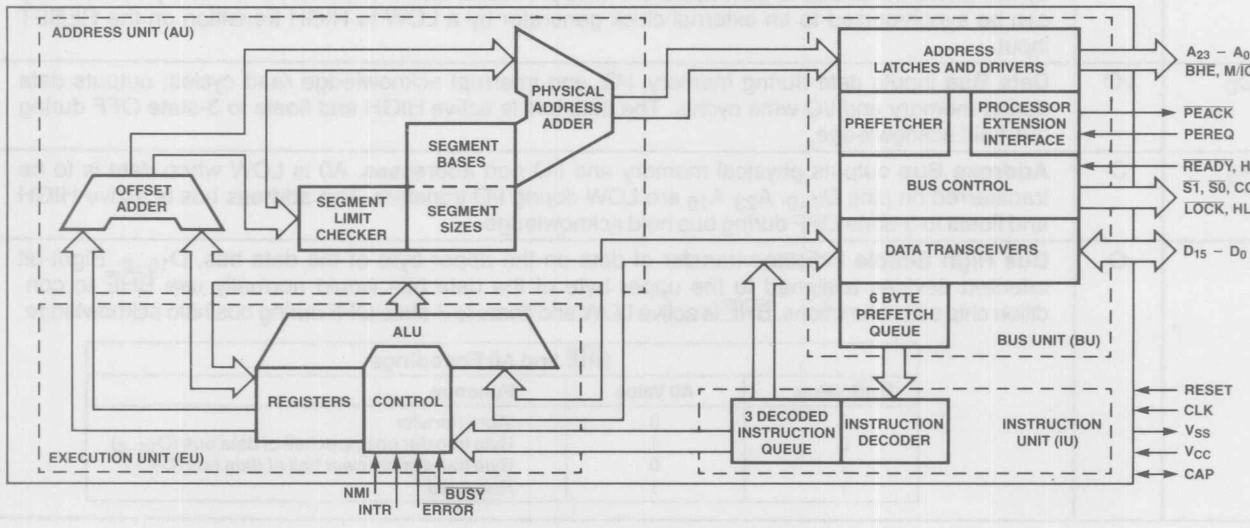
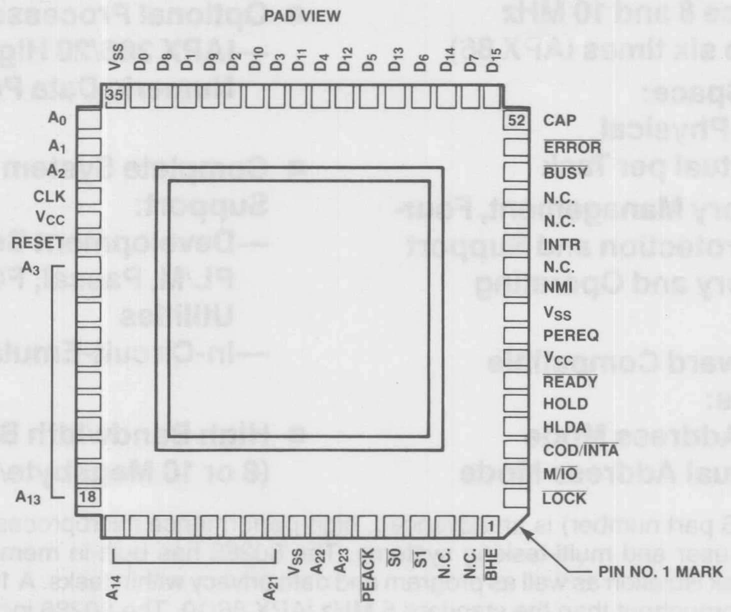


Figure 1. 80286 Internal Block Diagram

JANUARY 1982
ORDER NUMBER: 210253-001



NOTE: N.C. pads must not be connected.

Figure 2. 80286 Pin Configuration

Table 1. Pin Description

The following pin function descriptions are for the 80286 microprocessor:

Symbol	Type	Name and Function
CLK	I	**System Clock** provides the fundamental timing for iAPX 286 systems. It is a 16 MHz signal divided by two inside the 80286 to generate the 8 MHz processor clock. The internal divide-by-two circuitry can be synchronized to an external clock generator by a LOW to HIGH transition on the RESET input.
D_{15}-D_0	I/O	**Data Bus** inputs data during memory, I/O, and interrupt acknowledge read cycles; outputs data during memory and I/O write cycles. The data bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
A_{23}-A_0	O	**Address Bus** outputs physical memory and I/O port addresses. A0 is LOW when data is to be transferred on pins D_{7-0}. A_{23}-A_{16} are LOW during I/O transfers. The address bus is active HIGH and floats to 3-state OFF during bus hold acknowledge.
$\overline{BHE}$	O	**Bus High Enable** indicates transfer of data on the upper byte of the data bus, D_{15-8}. Eight-bit oriented devices assigned to the upper byte of the data bus would normally use $\overline{BHE}$ to condition chip select functions. $\overline{BHE}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.

$\overline{BHE}$ and A0 Encodings

$\overline{BHE}$ Value	A0 Value	Function
0	0	Word transfer
0	1	Byte transfer on upper half of data bus (D_{15-8})
1	0	Byte transfer on lower half of data bus (D_{7-0})
1	1	Reserved

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
$\overline{S1}$, $\overline{S0}$	O	**Bus Cycle Status** indicates initiation of a bus cycle and, along with M/$\overline{IO}$ and COD/$\overline{INTA}$, defines the type of bus cycle. The bus is in a T_S state whenever one or both are LOW. $\overline{S1}$ and $\overline{S0}$ are active LOW and float to 3-state OFF during bus hold acknowledge.

80286 Bus Cycle Status Definition

COD/$\overline{INTA}$	M/$\overline{IO}$	$\overline{S1}$	$\overline{S0}$	Bus cycle initiated
0 (LOW)	0	0	0	Interrupt acknowledge
0	0	0	1	Reserved
0	0	1	0	Reserved
0	0	1	1	None; not a status cycle
0	1	0	0	IF A1 = 1 then halt; else shutdown
0	1	0	1	Memory data read
0	1	1	0	Memory data write
0	1	1	1	None; not a status cycle
1 (HIGH)	0	0	0	Reserved
1	0	0	1	I/O read
1	0	1	0	I/O write
1	0	1	1	None; not a status cycle
1	1	0	0	Reserved
1	1	0	1	Memory instruction read
1	1	1	0	Reserved
1	1	1	1	None; not a status cycle

Symbol	Type	Name and Function
M/$\overline{IO}$	O	**Memory/IO Select** distinguishes memory access from I/O access. If HIGH during T_S, a memory cycle or a halt/shutdown cycle is in progress. If LOW, an I/O cycle or an interrupt acknowledge cycle is in progress. M/$\overline{IO}$ floats to 3-state OFF during bus hold acknowledge.
COD/$\overline{INTA}$	O	**Code/Interrupt Acknowledge** distinguishes instruction fetch cycles from memory data read cycles. Also distinguishes interrupt acknowledge cycles from I/O cycles. COD/$\overline{INTA}$ floats to 3-state OFF during bus hold acknowledge.
$\overline{LOCK}$	O	**Bus Lock** indicates that other system bus masters are not to gain control of the system bus following the current bus cycle. The $\overline{LOCK}$ signal may be activated explicitly by the "LOCK" instruction prefix or automatically by 80286 hardware during memory XCHG instructions, interrupt acknowledge, or descriptor table access. $\overline{LOCK}$ is active LOW and floats to 3-state OFF during bus hold acknowledge.
$\overline{READY}$	I	**Bus Ready** terminates a bus cycle. Bus cycles are extended without limit until terminated by $\overline{READY}$ LOW. $\overline{READY}$ is an active LOW synchronous input requiring setup and hold times relative to the system clock be met for correct operation. $\overline{READY}$ is ignored during bus hold acknowledge.
HOLD HLDA	I O	**Bus Hold Request and Hold Acknowledge** control ownership of the 80286 local bus. The HOLD input allows another local bus master to request control of the local bus. When control is granted, the 80286 will float its bus drivers to 3-state OFF and then activate HLDA, thus entering the bus hold acknowledge condition. The local bus will remain granted to the requesting master until HOLD becomes inactive which results in the 80286 deactivating HLDA and regaining control of the local bus. This terminates the bus hold acknowledge condition. HOLD may be asynchronous to the system clock. These signals are active HIGH.
INTR	I	**Interrupt Request** requests the 80286 to suspend its current program execution and service a pending external request. Interrupt requests are masked whenever the interrupt enable bit in the flag word is cleared. When the 80286 responds to an interrupt request, it performs two interrupt acknowledge bus cycles to read an 8-bit interrupt vector that identifies the source of the interrupt. To assure program interruption, INTR must remain active until the first interrupt acknowledge cycle is completed. INTR is sampled at the beginning of each processor cycle and must be active HIGH at least two processor cycles before the current instruction ends in order to interrupt before the next instruction. INTR is level sensitive, active HIGH, and may be asynchronous to the system clock.
NMI	I	**Non-maskable Interrupt Request** interrupts the 80286 with an internally supplied vector value of 2. No interrupt acknowledge cycles are performed. The interrupt enable bit in the 80286 flag word does not affect this input. The NMI input is active HIGH, may be asynchronous to the system clock, and is edge triggered after internal synchronization. For proper recognition, the input must have been previously LOW for at least four system clock cycles and remain HIGH for at least four system clock cycles.

Table 1. Pin Description (Cont.)

Symbol	Type	Name and Function
PEREQ $\overline{\text{PEACK}}$	I O	**Processor Extension Operand Request and Acknowledge** extend the memory management and protection capabilities of the 80286 to processor extensions. The PEREQ input requests the 80286 to perform a data operand transfer for a processor extension. The $\overline{\text{PEACK}}$ output signals the processor extension when the requested operand is being transferred. PEREQ is active HIGH and may be asynchronous to the system clock. $\overline{\text{PEACK}}$ is active LOW.
$\overline{\text{BUSY}}$ $\overline{\text{ERROR}}$	I I	**Processor Extension Busy and Error** indicate the operating condition of a processor extension to the 80286. An active $\overline{\text{BUSY}}$ input stops 80286 program execution on WAIT and some ESC instructions until $\overline{\text{BUSY}}$ becomes inactive (HIGH). The 80286 may be interrupted while waiting for $\overline{\text{BUSY}}$ to become inactive. An active $\overline{\text{ERROR}}$ input causes the 80286 to perform a processor extension interrupt when executing WAIT or some ESC instructions. These inputs are active LOW and may be asynchronous to the system clock.
RESET	I	**System Reset** clears the internal logic of the 80286 and is active HIGH. The 80286 may be re-initialized at any time with a LOW to HIGH transition on RESET which remains active for more than 16 system clock cycles. During RESET active, the output pins of the 80286 enter the state shown below:

80286 Pin State During Reset

Pin Value	Pin Names
1 (HIGH)	$\overline{\text{S0}}$, $\overline{\text{S1}}$, $\overline{\text{PEACK}}$, A23-A0, $\overline{\text{BHE}}$, $\overline{\text{LOCK}}$
0 (LOW)	M/$\overline{\text{IO}}$, COD/$\overline{\text{INTA}}$, HLDA
3-state OFF	D_{15}–D_0

Symbol	Type	Name and Function
RESET (cont.)		Operation of the 80286 begins after a HIGH to LOW transition on RESET. The HIGH to LOW transition of RESET must be synchronous to the system clock. Approximately 50 system clock cycles are required by the 80286 for internal initializations before the first bus cycle to fetch code from the power-on execution address is performed. A LOW to HIGH transition of RESET synchronous to the system clock, will begin a new processor cycle at the next HIGH to LOW transition of the system clock. The LOW to HIGH transition of RESET may be asynchronous to the system clock; however, in this case it can not be predetermined which phase of the processor clock will occur during the next system clock period. Synchronous LOW to HIGH transitions of RESET are only required for systems where the processor clock must be phase synchronous to another clock.
V_{SS}	I	**System Ground:** 0 VOLTS.
V_{CC}	I	**System Power:** +5 Volt Power Supply.
CAP	I	**Substrate Filter Capacitor:** a 0.047μf ± 20% 12V capacitor must be connected between this pin and ground. This capacitor filters the output of the internal substrate bias generator. A maximum DC leakage current of 1 μa is allowed through the capacitor. For correct operation of the 80286, the substrate bias generator must charge this capacitor to its operating voltage. The capacitor chargeup time is 5 milliseconds (max.) after V_{CC} and CLK reach their specified AC and DC parameters. RESET may be applied to prevent spurious activity by the CPU during this time. After this time, the 80286 processor clock can be phase synchronized to another clock by pulsing RESET LOW synchronous to the system clock.

FUNCTIONAL DESCRIPTION

Introduction

The 80286 is an advanced, high-performance microprocessor with specially optimized capabilities for multiple user and multi-tasking systems. Depending on the application, the 80286's performance is up to six times faster than the standard 5 MHz 8086's, while providing complete upward software compatibility with Intel's iAPX 86, 88, and 186 family of CPU's.

The 80286 operates in two modes: iAPX 86 real address mode and protected virtual address mode. Both modes execute a superset of the iAPX 86 and 88 instruction set.

In iAPX 86 real address mode programs use real addresses with up to one megabyte of address space. Programs use virtual addresses in protected virtual address mode, also called protected mode. In protected mode, the 80286 CPU automatically maps 1 gigabyte of virtual addresses per task into a 16 megabyte real address space. This mode also provides memory protection to isolate the operating system and ensure privacy of each tasks' programs and data. Both modes provide the same base instruction set, registers, and addressing modes.

The following Functional Description describes first, the base 80286 architecture common to both modes, second, iAPX 86 real address mode, and third, protected mode.

iAPX 286/10 BASE ARCHITECTURE

The iAPX 86, 88, 186, and 286 CPU family all contain the same basic set of registers, instructions, and addressing modes. The 80286 processor is upward compatible with the 8086, 8088, and 80186 CPU's.

Register Set

The 80286 base architecture has fifteen registers as shown in Figure 3. These registers are grouped into the following four categories:

General Registers: Eight 16-bit general purpose registers used to contain arithmetic and logical operands. Four of these (AX, BX, CX, and DX) can be used either in their entirety as 16-bit words or split into pairs of separate 8-bit registers.

Segment Registers: Four 16-bit special purpose registers select, at any given time, the segments of memory that are immediately addressable for code, stack, and data. (For usage, refer to Memory Organization.)

Base and Index Registers: Four of the general purpose registers may also be used to determine offset addresses of operands in memory. These registers may contain base addresses or indexes to particular locations within a segment. The addressing mode determines the specific registers used for operand address calculations.

Status and Control Registers: Three 16-bit special purpose registers record or control certain aspects of the 80286 processor state. These include the Instruction Pointer, which contains the offset address of the next sequential instruction to be executed.

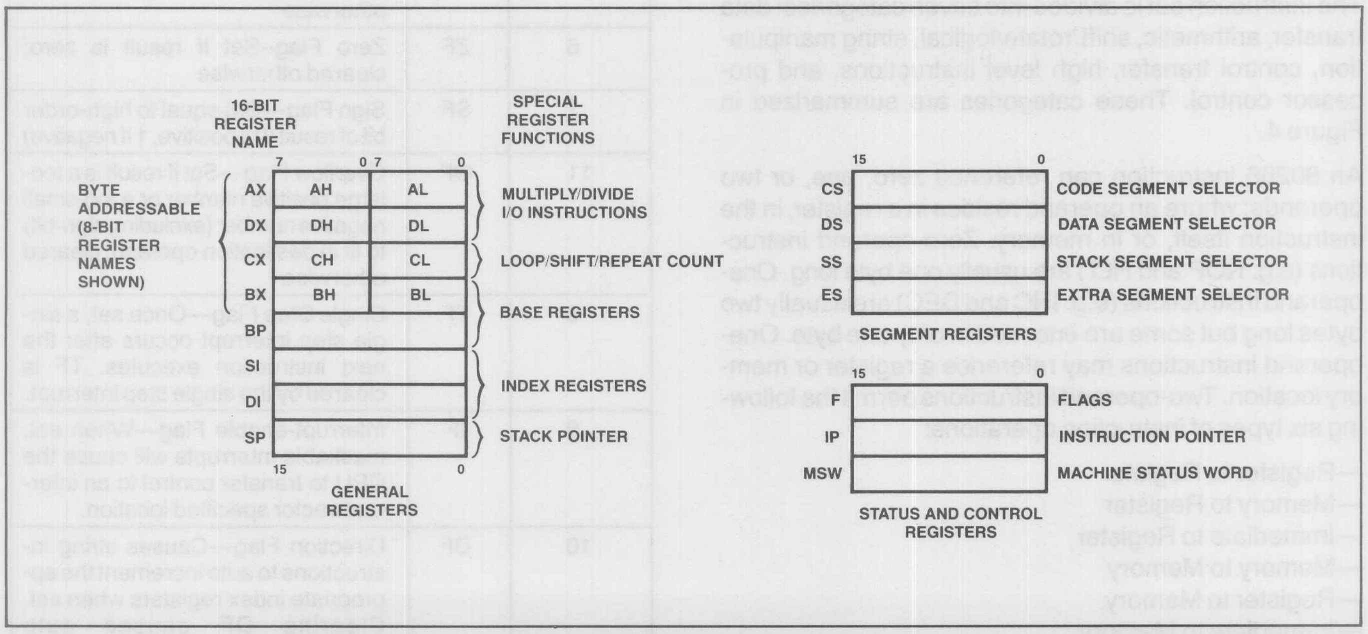


Figure 3. Register Set

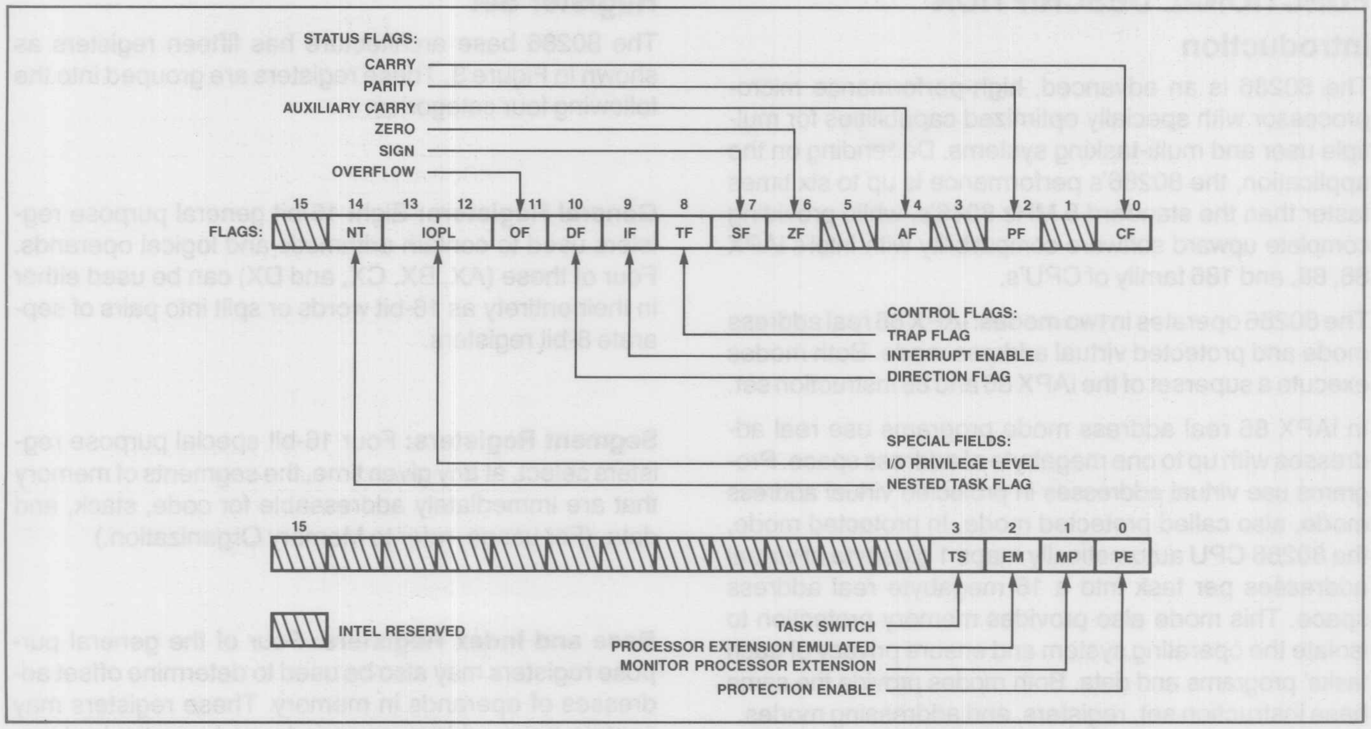


Figure 3a. Status and Control Register Bit Functions

Flags Word Description

The Flags word (Flags) records specific characteristics of the result of logical and arithmetic instructions (bits 0, 2, 4, 6, 7, and 11) and controls the operation of the 80286 within a given operating mode (bits 8 and 9). Flags is a 16-bit register. The function of the flag bits is given in Table 2.

Instruction Set

The instruction set is divided into seven categories: data transfer, arithmetic, shift/rotate/logical, string manipulation, control transfer, high level instructions, and processor control. These categories are summarized in Figure 4.

An 80286 instruction can reference zero, one, or two operands; where an operand resides in a register, in the instruction itself, or in memory. Zero-operand instructions (e.g. NOP and HLT) are usually one byte long. One-operand instructions (e.g. INC and DEC) are usually two bytes long but some are encoded in only one byte. One-operand instructions may reference a register or memory location. Two-operand instructions permit the following six types of instruction operations:

—Register to Register
—Memory to Register
—Immediate to Register
—Memory to Memory
—Register to Memory
—Immediate to Memory

Table 2. Flags Word Bit Functions

Bit Position	Name	Function
0	CF	Carry Flag—Set on high-order bit carry or borrow; cleared otherwise
2	PF	Parity Flag—Set if low-order 8 bits of result contain an even number of 1-bits; cleared otherwise
4	AF	Set on carry from or borrow to the low order four bits of AL; cleared otherwise
6	ZF	Zero Flag–Set if result is zero; cleared otherwise
7	SF	Sign Flag—Set equal to high-order bit of result (0 if positive, 1 if negative)
11	OF	Overflow Flag—Set if result is a too-large positive number or a too-small negative number (excluding sign-bit) to fit in destination operand; cleared otherwise
8	TF	Single Step Flag—Once set, a single step interrupt occurs after the next instruction executes. TF is cleared by the single step interrupt.
9	IF	Interrupt-enable Flag—When set, maskable interrupts will cause the CPU to transfer control to an interrupt vector specified location.
10	DF	Direction Flag—Causes string instructions to auto increment the appropriate index registers when set. Clearing DF causes auto decrement.

Two-operand instructions (e.g. MOV and ADD) are usually three to six bytes long. Memory to memory operations are provided by a special class of string instructions requiring one to three bytes. For detailed instruction formats and encodings refer to the instruction set summary at the end of this document.

	GENERAL PURPOSE
MOV	Move byte or word
PUSH	Push word onto stack
POP	Pop word off stack
PUSHA	Push all registers on stack
POPA	Pop all registers from stack
XCHG	Exchange byte or word
XLAT	Translate byte
	INPUT/OUTPUT
IN	Input byte or word
OUT	Output byte or word
	ADDRESS OBJECT
LEA	Load effective address
LDS	Load pointer using DS
LES	Load pointer using ES
	FLAG TRANSFER
LAHF	Load AH register from flags
SAHF	Store AH register in flags
PUSHF	Push flags onto stack
POPF	Pop flags off stack

Figure 4a. Data Transfer Instructions

	ADDITION
ADD	Add byte or word
ADC	Add byte or word with carry
INC	Increment byte or word by 1
AAA	ASCII adjust for addition
DAA	Decimal adjust for addition
	SUBTRACTION
SUB	Subtract byte or word
SBB	Subtract byte or word with borrow
DEC	Decrement byte or word by 1
NEG	Negate byte or word
CMP	Compare byte or word
AAS	ASCII adjust for subtraction
DAS	Decimal adjust for subtraction
	MULTIPLICATION
MUL	Multiply byte or word unsigned
IMUL	Integer multiply byte or word
AAM	ASCII adjust for multiply
	DIVISION
DIV	Divide byte or word unsigned
IDIV	Integer divide byte or word
AAD	ASCII adjust for division
CBW	Convert byte to word
CWD	Convert word to doubleword

Figure 4b. Arithmetic Instructions

MOVS	Move byte or word string
INS	Input bytes or word string
OUTS	Output bytes or word string
CMPS	Compare byte or word string
SCAS	Scan byte or word string
LODS	Load byte or word string
STOS	Store byte or word string
REP	Repeat
REPE/REPZ	Repeat while equal/zero
REPNE/REPNZ	Repeat while not equal/not zero

Figure 4c. String Instructions

	LOGICALS
NOT	"Not" byte or word
AND	"And" byte or word
OR	"Inclusive or" byte or word
XOR	"Exclusive or" byte or word
TEST	"Test" byte or word
	SHIFTS
SHL/SAL	Shift logical/arithmetic left byte or word
SHR	Shift logical right byte or word
SAR	Shift arithmetic right byte or word
	ROTATES
ROL	Rotate left byte or word
ROR	Rotate right byte or word
RCL	Rotate through carry left byte or word
RCR	Rotate through carry right byte or word

Figure 4d. Shift/Rotate/Logical Instructions

CONDITIONAL TRANSFERS	
JA/JNBE	Jump if above/not below nor equal
JAE/JNB	Jump if above or equal/not below
JB/JNAE	Jump if below/not above nor equal
JBE/JNA	Jump if below or equal/not above
JC	Jump if carry
JE/JZ	Jump if equal/zero
JG/JNLE	Jump if greater/not less nor equal
JGE/JNL	Jump if greater or equal/not less
JL/JNGE	Jump if less/not greater nor equal
JLE/JNG	Jump if less or equal/not greater
JNC	Jump if not carry
JNE/JNZ	Jump if not equal/not zero
JNO	Jump if not overflow
JNP/JPO	Jump if not parity/parity odd
JNS	Jump if not sign
JO	Jump if overflow
JP/JPE	Jump if parity/parity even
JS	Jump if sign

UNCONDITIONAL TRANSFERS	
CALL	Call procedure
RET	Return from procedure
JMP	Jump
ITERATION CONTROLS	
LOOP	Loop
LOOPE/LOOPZ	Loop if equal/zero
LOOPNE/LOOPNZ	Loop if not equal/not zero
JCXZ	Jump if register CX = 0
INTERRUPTS	
INT	Interrupt
INTO	Interrupt if overflow
IRET	Interrupt return

Figure 4e. Program Transfer Instructions

FLAG OPERATIONS	
STC	Set carry flag
CLC	Clear carry flag
CMC	Complement carry flag
STD	Set direction flag
CLD	Clear direction flag
STI	Set interrupt enable flag
CLI	Clear interrupt enable flag
EXTERNAL SYNCHRONIZATION	
HLT	Halt until interrupt or reset
WAIT	Wait for $\overline{\text{TEST}}$ pin active
ESC	Escape to extension processor
LOCK	Lock bus during next instruction
NO OPERATION	
NOP	No operation
EXECUTION ENVIRONMENT CONTROL	
LMSW	Load machine status word
SMSW	Store machine status word

Figure 4f. Processor Control Instructions

ENTER	Format stack for procedure entry
LEAVE	Restore stack for procedure exit
BOUND	Detects values outside prescribed range

Figure 4g. High Level Instructions

Memory Organization

Memory is organized as sets of variable length segments. Each segment is a linear contiguous sequence of up to 64K (2^{16}) 8-bit bytes. Memory is addressed using a two-component address (a pointer) that consists of a 16-bit segment selector, and a 16-bit offset. The segment selector indicates the desired segment in memory. The offset component indicates the desired byte address within the segment.

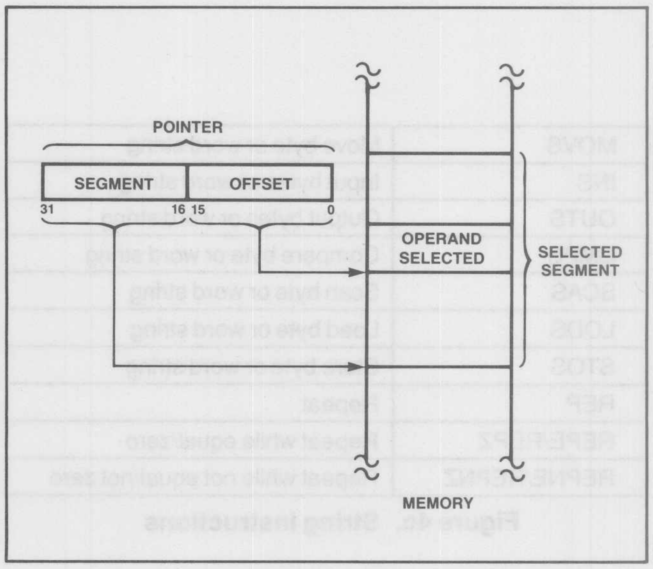


Figure 5. Two Component Address

Table 3. Segment Register Selection Rules

Memory Reference Needed	Segment Register Used	Implicit Segment Selection Rule
Instructions	Code (CS)	Automatic with instruction prefetch
Stack	Stack (SS)	All stack pushes and pops. Any memory reference which uses BP as a base register.
Local Data	Data (DS)	All data references except when relative to stack or string destination
External (Global) Data	Extra (ES)	Alternate data segment and destination of string operation

All instructions that address operands in memory must specify the segment and the offset. For speed and compact instruction encoding, segment selectors are usually stored in the high speed segment registers. An instruction need specify only the desired segment register and an offset in order to address a memory operand.

Most instructions need not explicitly specify which segment register is used. The correct segment register is automatically chosen according to the rules of Table 3. These rules follow the way programs are written (see Figure 6) as independent modules that require areas for code and data, a stack, and access to external data areas.

Special segment override instruction prefixes allow the implicit segment register selection rules to be overridden for special cases. The stack, data, and extra segments may coincide for simple programs. To access operands that do not reside in one of the four immediately available segments, either a full 32-bit pointer can be used or a new segment selector must be loaded.

Addressing Modes

The 80286 provides a total of eight addressing modes for instructions to specify operands. Two addressing modes are provided for instructions that operate on register or immediate operands:

Register Operand Mode: The operand is located in one of the 8 or 16-bit general registers.

Immediate Operand Mode. The operand is included in the instruction.

Six modes are provided to specify the location of an operand in a memory segment. A memory operand address consists of two 16-bit components: segment selector and offset. The segment selector is supplied by a segment register either implicitly chosen by the addressing mode or explicitly chosen by a segment override prefix. The offset is calculated by summing any combination of the following three address elements:

the **displacement** (an 8 or 16-bit immediate value contained in the instruction)

the **base** (contents of either the BX or BP base registers)

the **index** (contents of either the SI or DI index registers)

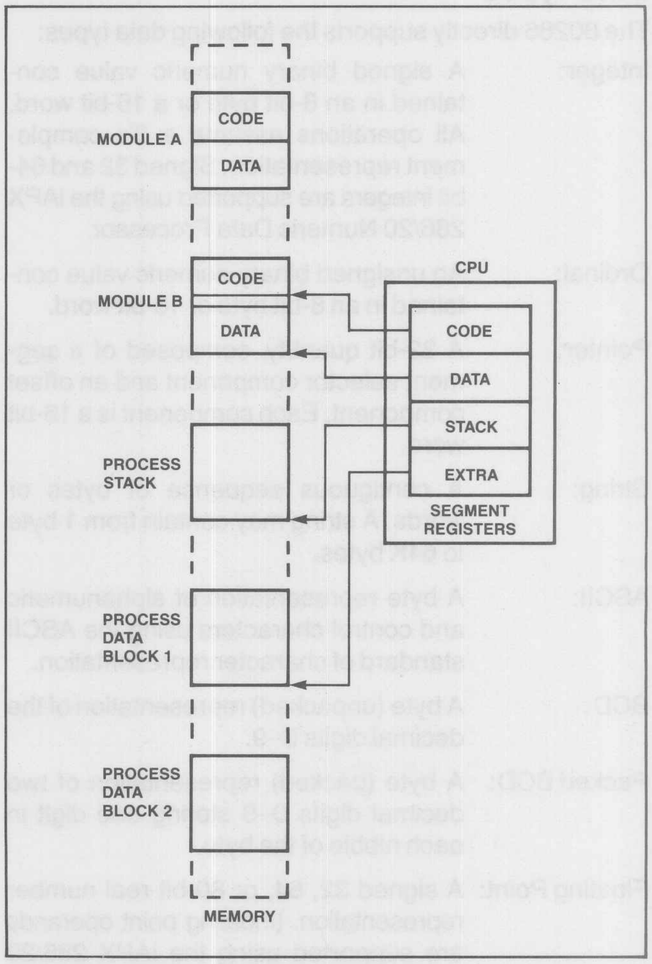


Figure 6. Segmented Memory Helps Structure Software

Any carry out from the 16-bit addition is ignored. Eight-bit displacements are sign extended to 16-bit values.

Combinations of these three address elements define the six memory addressing modes, described below.

Direct Mode: The operand's offset is contained in the instruction as an 8 or 16-bit displacement element.

Register Indirect Mode: The operand's offset is in one of the registers SI, DI, BX, or BP.

Based Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of a base register (BX or BP).

Indexed Mode: The operand's offset is the sum of an 8 or 16-bit displacement and the contents of an index register (SI or DI).

Based Indexed Mode: The operand's offset is the sum of the contents of a base register and an index register.

Based Indexed Mode with Displacement: The operand's offset is the sum of a base register's contents, an index register's contents, and an 8 or 16-bit displacement.

Data Types

The 80286 directly supports the following data types:

Integer:	A signed binary numeric value contained in an 8-bit byte or a 16-bit word. All operations assume a 2's complement representation. Signed 32 and 64-bit integers are supported using the iAPX 286/20 Numeric Data Processor.
Ordinal:	An unsigned binary numeric value contained in an 8-bit byte or 16-bit word.
Pointer:	A 32-bit quantity, composed of a segment selector component and an offset component. Each component is a 16-bit word.
String:	A contiguous sequence of bytes or words. A string may contain from 1 byte to 64K bytes.
ASCII:	A byte representation of alphanumeric and control characters using the ASCII standard of character representation.
BCD:	A byte (unpacked) representation of the decimal digits 0–9.
Packed BCD:	A byte (packed) representation of two decimal digits 0–9 storing one digit in each nibble of the byte.
Floating Point:	A signed 32, 64, or 80-bit real number representation. (Floating point operands are supported using the iAPX 286/20 Numeric Processor configuration.)

Figure 7 graphically represents the data types supported by the iAPX 286.

I/O Space

The I/O space consists of 64K 8-bit or 32K 16-bit ports. I/O instructions address the I/O space with either an 8-bit port address, specified in the instruction, or a 16-bit port address in the DX register. 8-bit port addresses are zero extended such that A_{15}–A_8 are LOW. I/O port addresses 00F8(H) through 00FF(H) are reserved.

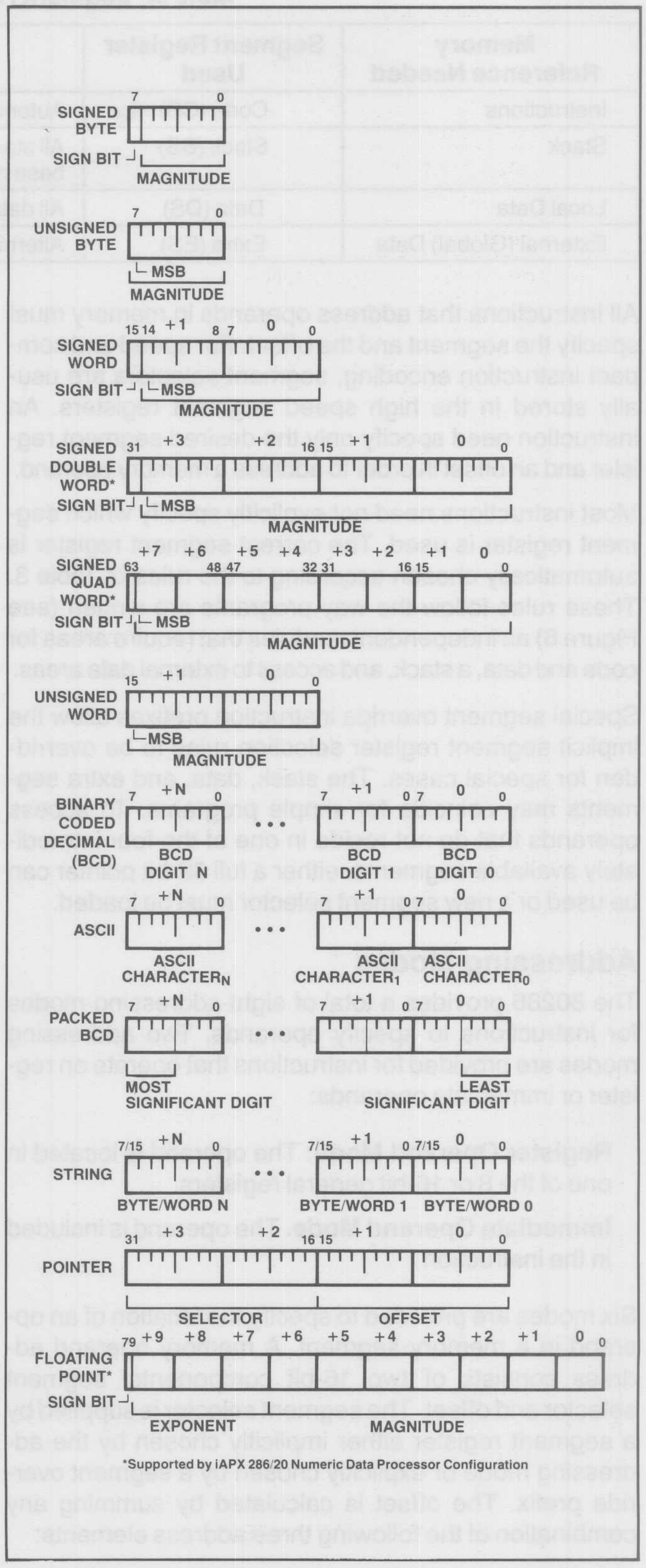


Figure 7. iAPX 286 Supported Data Types

Table 4. Interrupt Vector Assignments

Function	Interrupt Number	Related Instructions	Return Address Before Instruction Causing Exception?
Divide error exception	0	DIV, IDIV	Yes
Single step interrupt	1	All	
NMI interrupt	2	All	
Breakpoint interrupt	3	INT	
INTO detected overflow exception	4	INTO	No
BOUND range exceeded exception	5	BOUND	Yes
Invalid opcode exception	6	Any undefined opcode	Yes
Processor extension not available exception	7	ESC or WAIT with	Yes
Reserved	8–15		
Processor extension error interrupt	16	ESC or WAIT	
Reserved	17–31		
User defined	32–255		

Interrupts

An interrupt transfers execution to a new program location. The old program address (CS:IP) and machine state (Flags) are saved on the stack to allow resumption of the interrupted program. Interrupts fall into three classes: hardware initiated, INT instructions, and instruction exceptions. Hardware initiated interrupts occur in response to an external input and are classified as non-maskable or maskable. Programs may cause an interrupt with an INT instruction. Instruction exceptions occur when an unusual condition, which prevents further instruction processing, is detected while attempting to execute an instruction. The return address from an exception will always point at the instruction causing the exception and include any leading instruction prefixes.

A table containing up to 256 pointers defines the proper interrupt service routine for each interrupt. Interrupts 0–31, some of which are used for instruction exceptions, are reserved. For each interrupt, an 8-bit vector must be supplied to the 80286 which identifies the appropriate table entry. Exceptions supply the interrupt vector internally. INT instructions contain or imply the vector and allow access to all 256 interrupts. Maskable hardware initiated interrupts supply the 8-bit vector to the CPU during an interrupt acknowledge bus sequence. Non-maskable hardware interrupts use a predefined internally supplied vector.

MASKABLE INTERRUPT (INTR)

The 80286 provides a maskable hardware interrupt request pin, INTR. Software enables this input by setting the interrupt flag bit (IF) in the flag word. All 224 user-defined interrupt sources can share this input, yet they can retain separate interrupt handlers. An 8-bit vector read by the CPU during the interrupt acknowledge sequence (discussed in System Interface section) identifies the source of the interrupt.

Further maskable interrupts are disabled while servicing an interrupt by resetting the IF but as part of the response to an interrupt or exception. The saved flag word will reflect the enable status of the processor prior to the interrupt. Until the flag word is restored to the flag register, the interrupt flag will be zero unless specifically set. The interrupt return instruction includes restoring the flag word, thereby restoring the original status of IF.

NON-MASKABLE INTERRUPT REQUEST (NMI)

A non-maskable interrupt input (NMI) is also provided. NMI has higher priority than INTR. A typical use of NMI would be to activate a power failure routine. The activation of this input causes an interrupt with an internally supplied vector value of 2. No external interrupt acknowledge sequence is performed.

While executing the NMI servicing procedure, the 80286 will service neither further NMI requests, INTR requests, nor the processor extension segment overrun interrupt until an interrupt return (IRET) instruction is executed or the CPU is reset. If NMI occurs while currently servicing an NMI, its presence will be saved for servicing after executing the first IRET instruction. IF is cleared at the beginning of an NMI interrupt to inhibit INTR interrupts.

SINGLE STEP INTERRUPT

The 80286 has an internal interrupt that allows programs to execute one instruction at a time. It is called the single step interrupt and is controlled by the single step flag bit (TF) in the flag word. Once this bit is set, an internal single step interrupt will occur after the next instruction has been executed. The interrupt clears the TF bit and uses an internally supplied vector of 1. The IRET instruction is used to set the TF bit and transfer control to the next instruction to be single stepped.

Interrupt Priorities

When simultaneous interrupt requests occur, they are processed in a fixed order as shown in Table 5. Interrupt processing involves saving the flags, return address, and setting CS:IP to point at the first instruction of the interrupt handler. If other interrupts remain enabled they are processed before the first instruction of the current interrupt handler is executed. The last interrupt processed is therefore the first one serviced.

Table 5. Interrupt Processing Order

Order	Interrupt
1	INT instruction or exception
2	Single step
3	NMI
4	Processor extension segment overrun
5	INTR

Initialization and Processor Reset

Processor initialization or start up is accomplished by driving the RESET input pin HIGH. RESET forces the 80286 to terminate all execution and local bus activity. No instruction or bus activity will occur as long as RESET is active. After RESET becomes inactive and an internal processing interval elapses, the 80286 begins execution in real address mode with the instruction at physical location FFFFF0(H). RESET also sets some registers to predefined values as shown as shown in Table 6.

Table 6. 80286 Initial Register State after RESET

Flag word	0002(H)
Machine Status Word	FFF0(H)
Instruction pointer	FFF0(H)
Code segment	F000(H)
Data segment	0000(H)
Extra segment	0000(H)
Stack segment	0000(H)

Machine Status Word Description

The machine status word (MSW) records when a task switch takes place and controls the operating mode of the 80286. It is a 16-bit register of which the lower four bits are used. One bit places the CPU into protected mode, while the other three bits, as shown in Table 7, control the processor extension interface. After RESET, this register contains FFF0(H) which places the 80286 in iAPX 86 real address mode.

Table 7. MSW Bit Functions

Bit Position	Name	Function
0	PE	Protected mode enable places the 80286 into protected mode and can not be cleared except by RESET.
1	MP	Monitor processor extension allows WAIT instructions to cause a processor extension not present exception (number 7).
2	EM	Emulate processor extension causes a processor extension not present exception (number 7) on ESC instructions to allow emulating a processor extension.
3	TS	Task switched indicates the next instruction using a processor extension will cause exception 7, allowing software to test whether the current processor extension context belongs to the current task.

The LMSW and SMSW instructions can load and store the MSW in real address mode. The recommended use of TS, EM, and MP is shown in Table 8.

Table 8. Recommended MSW Encodings For Processor Extension Control

TS	MP	EM	Recommended Use	Instructions Causing Exception
0	0	0	iAPX 86 real address mode only. Initial encoding after RESET. iAPX 286 operation is identical to iAPX 86, 88.	None
0	0	1	No processor extension is available. Software will emulate its function.	ESC
1	0	1	No processor extension is available. Software will emulate its function. The current processor extension context may belong to another task.	ESC
0	1	0	A processor extension exists.	None
1	1	0	A processor extension exists. The current processor extension context may belong to another task. The exception on WAIT allows software to test for an error pending from a previous processor extension operation.	ESC or WAIT

Halt

The HLT instruction stops program execution and prevents the CPU from using the local bus until restarted. Either NMI, INTR with IF = 1, or RESET will force the 80286 out of halt. If interrupted, the saved CS:IP will point to the next instruction after the HLT.

iAPX 86 REAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in real address mode. In real address mode the 80286 is object code compatible with 8086 and 8088 software. The real address mode architecture (registers and addressing modes) is exactly as described in the iAPX 286/10 Base Architecture section of this Functional Description.

Memory Size

Physical memory is a contiguous array of up to 1,048,576 bytes (one megabyte) addressed by pins A_0 through A_{19} and $\overline{BHE}$. A_{20} through A_{23} are ignored.

Memory Addressing

In real address mode the processor generates 20-bit physical addresses directly from a 20-bit segment base address and a 16-bit offset.

The selector portion of a pointer is interpreted as the upper 16 bits of a 20-bit segment address. The lower four bits of the 20-bit segment address are always zero. Segment addresses, therefore, begin on multiples of 16 bytes. See Figure 8 for a graphic representation of address formation.

All segments in real address mode are 64K bytes in size and may be read, written, or executed. An exception or interrupt can occur if data operands or instructions attempt to wrap around the end of a segment (e.g. a word with its low order byte at offset FFFF(H) and its high order byte at offset 0000(H)). If, in real address mode, the information contained in a segment does not use the full 64K bytes, the unused end of the segment may be overlayed by another segment to reduce physical memory requirements.

Reserved Memory Locations

The 80286 reserves two fixed areas of memory in real address mode (see Figure 9); system initialization area and interrupt table area. Locations from addresses FFFF0(H) thorugh FFFFF(H) are reserved for system initialization. Initial execution begins at location FFFF0(H). Locations 00000(H) through 003FF(H) are reserved for interrupt vectors.

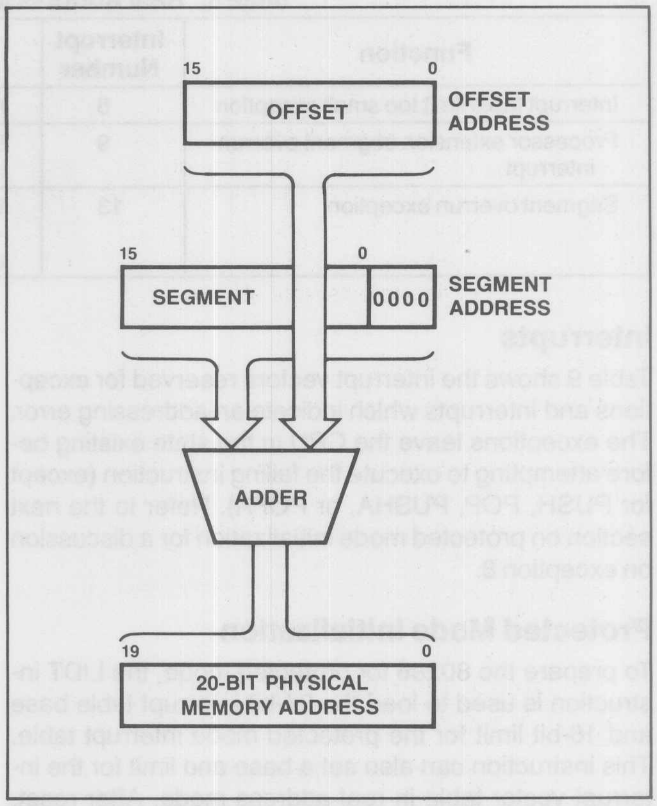


Figure 8. iAPX 86 Real Address Mode Address Calculation

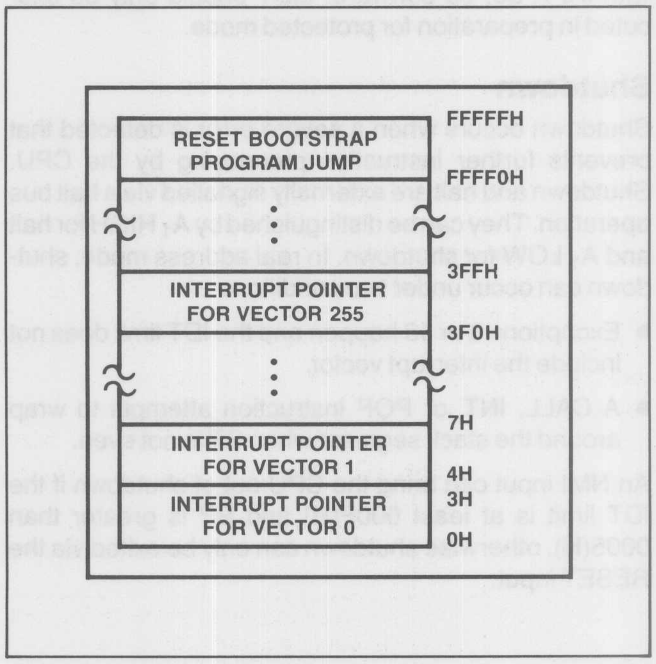

FFFFFH
RESET BOOTSTRAP PROGRAM JUMP
FFFF0H
3FFH
INTERRUPT POINTER FOR VECTOR 255
3F0H
7H
INTERRUPT POINTER FOR VECTOR 1
4H
3H
INTERRUPT POINTER FOR VECTOR 0
0H

Figure 9. iAPX 86 Real Address Mode Initially Reserved Memory Locations

Table 9. Real Address Mode Addressing Interrupts

Function	Interrupt Number	Related Instructions	Return Address Before Instruction?
Interrupt table limit too small exception	8	INT vector is not within table limit	Yes
Processor extension segment overrun interrupt	9	ESC with memory operand extending beyond offset FFFF(H)	No
Segment overrun exception	13	Word memory reference with offset = FFFF(H) or an attempt to execute past the end of a segment	Yes

Interrupts

Table 9 shows the interrupt vectors reserved for exceptions and interrupts which indicate an addressing error. The exceptions leave the CPU in the state existing before attempting to execute the failing instruction (except for PUSH, POP, PUSHA, or POPA). Refer to the next section on protected mode initialization for a discussion on exception 8.

Protected Mode Initialization

To prepare the 80286 for protected mode, the LIDT instruction is used to load the 24-bit interrupt table base and 16-bit limit for the protected mode interrupt table. This instruction can also set a base and limit for the interrupt vector table in real address mode. After reset, the interrupt table base is initialized to 000000(H) and its size set to 03FF(H). These values are compatible with iAPX 86, 88 software. LIDT should only be executed in preparation for protected mode.

Shutdown

Shutdown occurs when a severe error is detected that prevents further instruction processing by the CPU. Shutdown and halt are externally signalled via a halt bus operation. They can be distinguished by A_1 HIGH for halt and A_1 LOW for shutdown. In real address mode, shutdown can occur under two conditions:

- Exceptions 8 or 13 happen and the IDT limit does not include the interrupt vector.
- A CALL, INT, or POP instruction attempts to wrap around the stack segment when SP is not even.

An NMI input can bring the CPU out of shutdown if the IDT limit is at least 000F(H) and SP is greater than 0005(H), otherwise shutdown can only be exited via the RESET input.

PROTECTED VIRTUAL ADDRESS MODE

The 80286 executes a fully upward-compatible superset of the 8086 instruction set in protected virtual address mode (protected mode). Protected mode also provides memory management and protection mechanisms and associated instructions.

The 80286 enters protected virtual address mode from real address mode by setting the PE (Protection Enable) bit of the machine status word with the Load Machine Status Word (LMSW) instruction. Protected mode offers extended physical and virtual memory address space, memory protection mechanisms, and new operations to support operating systems and virtual memory.

All registers, instructions, and addressing modes described in the iAPX 286/10 Base Architecture section of this Functional Description remain the same. Programs for the iAPX 86, 88, 186, and real address mode 80286 can be run in protected mode; however, embedded constants for segment selectors are different.

Memory Size

The protected mode 80286 provides a 1 gigabyte virtual address space per task mapped into a 16 megabyte physical address space defined by the address pins A_{23}–A_0 and $\overline{BHE}$. The virtual address space may be larger than the physical address space since any use of an address that does not map to a physical memory location will cause a restartable exception.

Memory Addressing

As in real address mode, protected mode uses 32-bit pointers, consisting of 16-bit selector and offset components. The selector, however, specifies an index into a memory resident table rather than the upper 16-bits of a real memory address. The 24-bit base address of the desired segment is obtained from the tables in memory. The 16-bit offset is added to the segment base address to form the physical address as shown in Figure 10. The tables are automatically referenced by the CPU whenever a segment register is loaded with a selector. All iAPX 286 instructions which load a segment register will reference the memory based tables without additional software. The memory based tables contain 8 byte values called descriptors.

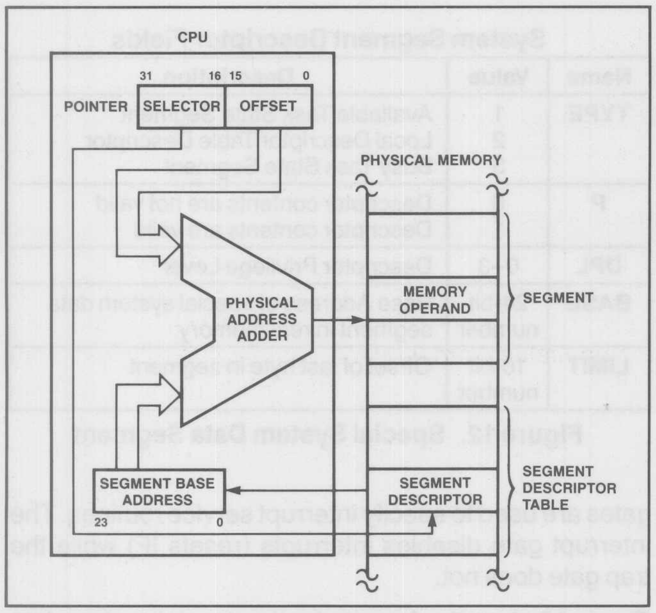


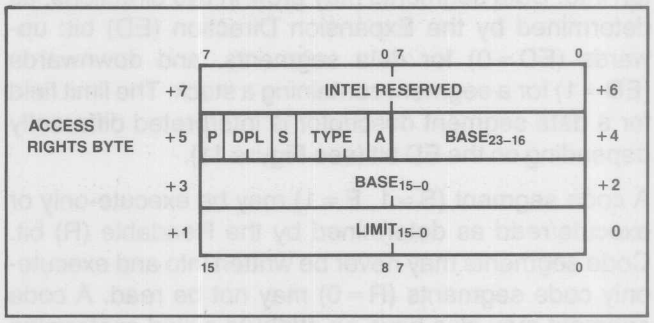
Figure 10. Protected Mode Memory Addressing

DESCRIPTORS

Descriptors define the use of memory. Special types of descriptors also define new functions for transfer of control and task switching. The 80286 has segment descriptors for code, stack and data segments, and system control descriptors for special system data segments and control transfer operations. Descriptor accesses are performed as locked bus operations to assure descriptor integrity in multi-processor systems.

CODE AND DATA SEGMENT DESCRIPTORS

Besides segment base addresses, code and data descriptors contain other segment attributes including segment size (1 to 64K bytes), access rights (read only, read/write, execute only, and execute/read), and presence in memory (for virtual memory systems) (See Figure 11). Any segment usage violating a segment attribute indicated by the segment descriptor will prevent the memory cycle and cause an exception or interrupt.



Access Rights Byte Definition

	Bit Position	Name	Function	
	7	Present (P)	P = 1 Segment is mapped into physical memory.	
			P = 0 No mapping to physical memory exists, base and limit are not used.	
	6–5	Descriptor Privilege Level (DPL)	Segment privilege attribute used in privilege tests.	
	4	Segment Descriptor (S)	S = 1 Code or Data segment descriptor	
			S = 0 Non-segment descriptor	
Type Field Definition	3	Executable (E)	E = 0 Data segment descriptor type is:	Data Segment
	2	Expansion Direction (ED)	ED = 0 Grow up segment, offsets must be ≤ limit.	
			ED = 1 Grow down segment, offsets must be > limit.	
	1	Writeable (W)	W = 0 Data segment may not be written into.	
			W = 1 Data segment may be written into.	
	3	Executable (E)	E = 1 Code Segment Descriptor type is:	Code Segment
	2	Conforming (C)	C = 0 Code segment may only be executed when CPL ≥ DPL.	
	1	Readable (R)	R = 0 Code segment may not be read.	
			R = 1 Code segment may be read.	
	0	Accessed (A)	A = 0 Segment has not been accessed.	
			A = 1 Segment selector has been loaded into segment register or used by selector test instructions.	

Figure 11. Code and Data Segment Descriptors

Code and data are stored in two types of segments: code segments and data segments. Both types are identified and defined by segment descriptors. Code segments are identified by the executable (E) bit set to 1 in the descriptor access rights byte. The access rights byte of both code and data segment descriptor types have three fields in common: present (P) bit, Descriptor Privilege Level (DPL), and accessed (A) bit. If P = 0, any attempted use of this segment will cause a not-present exception. DPL specifies the privilege level of the segment descriptor. DPL effects when the descriptor may be used by a task (refer to privilege discussion below). The A bit shows whether the segment has been previously accessed for usage profiling, a necessity for virtual memory systems. The CPU will always set this bit when accessing the descriptor.

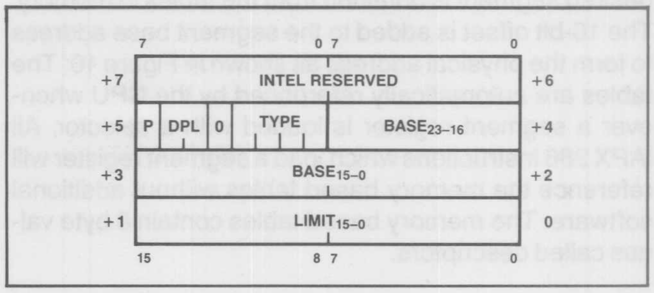
Data segments (S = 1, E = 0) may be either read-only or read-write as controlled by the W bit of the access rights byte. Read-only (W = 0) data segments may not be written into. Data segments may grow in two directions, as determined by the Expansion Direction (ED) bit: upwards (ED = 0) for data segments, and downwards (ED = 1) for a segment containing a stack. The limit field for a data segment descriptor is interpreted differently depending on the ED bit (see Figure 11).

A code segment (S = 1, E = 1) may be execute-only or execute/read as determined by the Readable (R) bit. Code segments may never be written into and execute-only code segments (R = 0) may not be read. A code segment may also have an attribute called conforming (C). A conforming code segment may be shared by programs that execute at different privilege levels. The DPL of a conforming code segment defines the range of privilege levels at which the segment may be executed (refer to privilege discussion below).

SYSTEM CONTROL DESCRIPTORS

In addition to code and data segment descriptors, the protected mode 80286 defines system control descriptors. These descriptors define special system data segments and control transfer mechanisms in the protected environment. The special system data segment descriptors define segments which contain tables of descriptors (Local Descriptor Table Descriptor) and segments which contain the execution state of a task (Task State Segment Descriptor).

The control transfer descriptors are call gates, task gates, interrupt gates and trap gates. Gates provide a level of indirection between the source and destination of the control transfer. This indirection allows the CPU to automatically perform protection checks and control the entry point of the destination. Call gates are used to change privilege levels (see Privilege), task gates are used to perform a task switch, and interrupt and trap gates are used to specify interrupt service routines. The interrupt gate disables interrupts (resets IF) while the trap gate does not.



System Segment Descriptor Fields

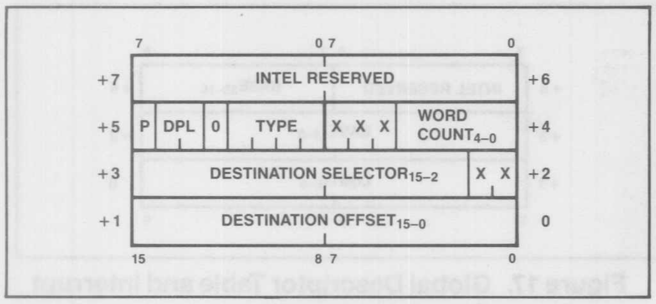
Name	Value	Description
TYPE	1 2 3	Available Task State Segment Local Descriptor Table Descriptor Busy Task State Segment
P	0 1	Descriptor contents are not valid Descriptor contents are valid
DPL	0–3	Descriptor Privilege Level
BASE	24-bit number	Base Address of special system data segment in real memory
LIMIT	16-bit number	Offset of last byte in segment

Figure 12. Special System Data Segment

Figure 12 gives the formats for the special system data segment descriptors. The descriptors contain a 24-bit base address of the segment and a 16-bit limit. The access byte defines the type of descriptor, its state and privilege level. The descriptor contents are valid and the segment is in physical memory if P = 1. If P = 0, the segment is not valid. The DPL field is only used in Task State Segment descriptors and indicates the privilege level at which the descriptor may be used (see Privilege). Since the Local Descriptor Table descriptor may only be used by a special privileged instruction, the DPL field is not used. Bit 4 of the access byte is 0 to indicate that it is a system control descriptor. The type field specifies the descriptor type as indicated in Figure 12.

Figure 13 shows the format of the gate descriptors. The descriptor contains a destination pointer that points to the descriptor of the target segment and the entry point offset. The destination selector in an interrupt gate, trap gate, and call gate must refer to a code segment descriptor. These gate descriptors contain the entry point to prevent a program from constructing and using an illegal entry point. Task gates may only refer to a task state segment. Since task gates invoke a task switch, the destination offset is not used in the task gate.

Exception 13 is generated when the gate is used if a destination selector does not refer to the correct de-



Gate Descriptor Fields

Name	Value	Description
TYPE	4 5 6 7	–Call Gate –Task Gate –Interrupt Gate –Trap Gate
P	0 1	–Descriptor Contents are not valid –Descriptor Contents are valid
DPL	0–3	Descriptor Privilege Level
WORD COUNT	0–31	Number of words to copy from callers stack to called procedures stack. Only used with call gate.
DESTINATION SELECTOR	16-bit selector	Selector to the target code segment (Call, Interrupt or Trap Gate) Selector to the target task state segment (Task Gate)
DESTINATION OFFSET	16-bit offset	Entry point within the target code segment

Figure 13. Gate Descriptor Format

scriptor type. The word count field is used in the call gate descriptor to indicate the number of parameters (0–31 words) to be automatically copied from the caller's stack to the stack of the called routine when a control transfer changes privilege levels. The word count field is not used by any other gate descriptor.

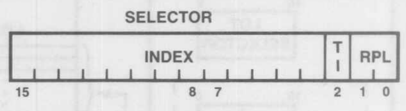
The access byte format is the same for all gate descriptors. P = 1 indicates that the gate contents are valid. P = 0 indicates the contents are not valid and causes exception 11 if referenced. DPL is the descriptor privilege level and specifies when this descriptor may be used by a task (refer to privilege discussion below). Bit 4 must equal 0 to indicate a system control descriptor. The type field specifies the descriptor type as indicated in Figure 13.

SEGMENT DESCRIPTOR CACHE REGISTERS

A segment descriptor cache register is assigned to each of the four segment registers (CS, SS, DS, ES). Segment descriptors are automatically loaded (cached) into a segment descriptor cache register (Figure 14) whenever the associated segment register is loaded with a selector. Only segment descriptors may be loaded into segment descriptor cache registers. Once loaded, all references to that segment of memory use the cached descriptor information instead of reaccessing memory. The descriptor cache registers are not visible to programs. No instructions exist to store their contents. They only change when a segment register is loaded.

SELECTOR FIELDS

A protected mode selector has three fields: descriptor entry index, local or global descriptor table indicator (TI), and selector privilege (RPL) as shown in Figure 15. These fields select one of two memory based tables of descriptors, select the appropriate table entry and allow high-speed testing of the selector's privilege attribute (refer to privilege discussion below).



BITS	NAME	FUNCTION
1–0	REQUESTED PRIVILEGE LEVEL (RPL)	INDICATES SELECTOR PRIVILEGE LEVEL DESIRED

2	TABLE INDICATOR (TI)	TI = 0 USE GLOBAL DESCRIPTOR TABLE (GDT)
		TI = 1 USE LOCAL DESCRIPTOR TABLE (LDT)

15–3	INDEX	SELECT DESCRIPTOR ENTRY IN TABLE

Figure 15. Selector Fields

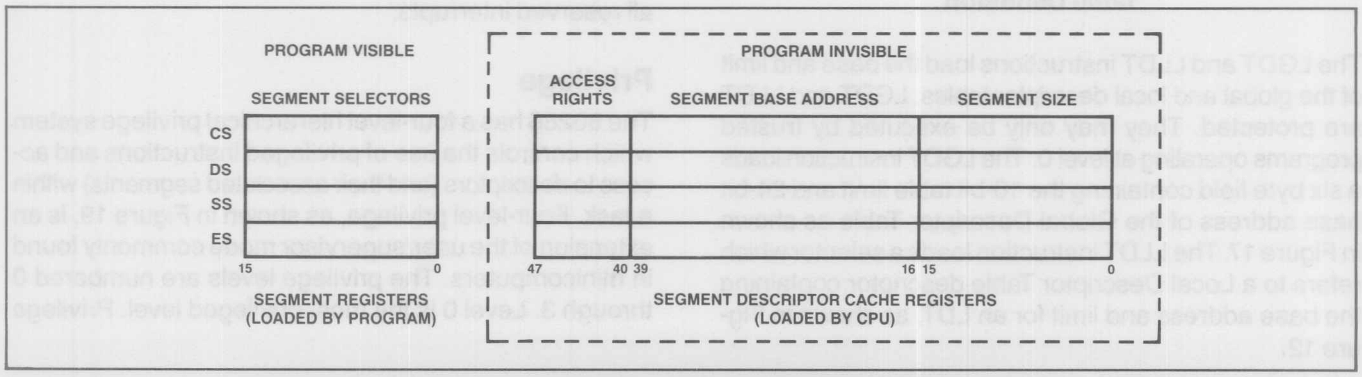


Figure 14. Descriptor Cache Registers

LOCAL AND GLOBAL DESCRIPTOR TABLES

Two tables of descriptors, called descriptor tables, contain all descriptors accessible by a task at any given time. A descriptor table is a linear array of up to 8192 descriptors. The upper 13 bits of the selector value are an index into a descriptor table. Each table has a 24-bit base register to locate the descriptor table in physical memory and a 16-bit limit register that confine descriptor access to the defined limits of the table as shown in Figure 16. A restartable exception (13) will occur if an attempt is made to reference a descriptor outside the table limits.

One table, called the Global Descriptor Table (GDT), contains descriptors available to all tasks. The other table, called the Local Descriptor Table (LDT), contains descriptors that can be private to a task. Each task may have its own private LDT. The GDT may contain all descriptor types except interrupt and trap descriptors. The LDT may contain only segment, task gate, and call gate descriptors. A segment cannot be accessed by a task if its segment descriptor does not exist in either descriptor table at the time of access.

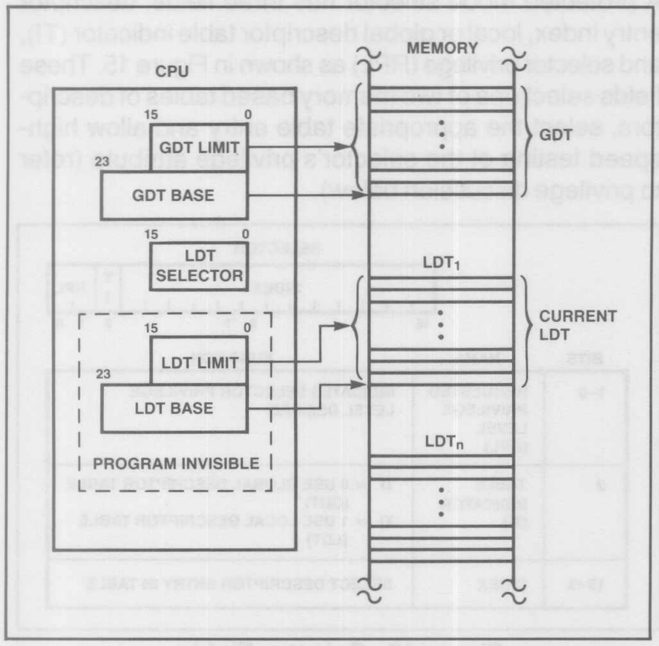


Figure 16. Local and Global Descriptor Table Definition

The LGDT and LLDT instructions load the base and limit of the global and local descriptor tables. LGDT and LLDT are protected. They may only be executed by trusted programs operating at level 0. The LGDT instruction loads a six byte field containing the 16-bit table limit and 24-bit base address of the Global Descriptor Table as shown in Figure 17. The LLDT instruction loads a selector which refers to a Local Descriptor Table descriptor containing the base address and limit for an LDT, as shown in Figure 12.

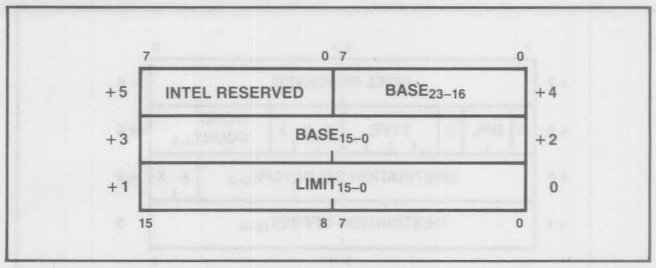


Figure 17. Global Descriptor Table and Interrupt Descriptor Data Type

INTERRUPT DESCRIPTOR TABLE

The protected mode 80286 has a third descriptor table, called the Interrupt Descriptor Table (IDT) (see Figure 18), used to define up to 256 interrupts. It may contain only task gates, interrupt gates and trap gates. The IDT (Interrupt Descriptor Table) has a 24-bit base and 16-bit limit register in the CPU. The protected LIDT instruction loads these registers with a six byte value of identical form to that of the LGDT instruction (see Figure 17 and Protected Mode Initialization).

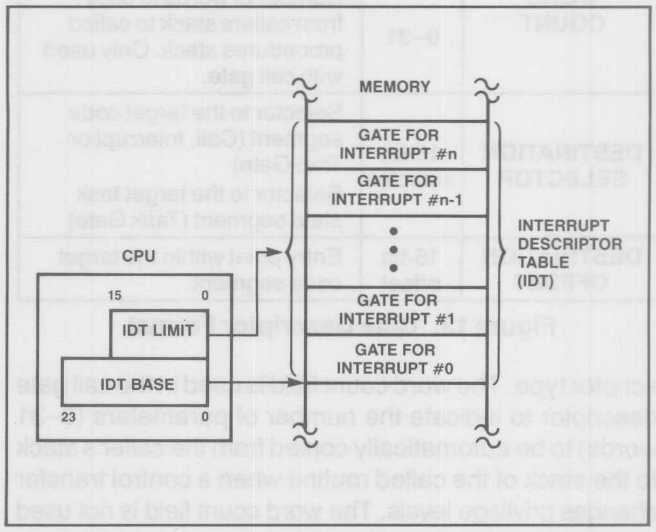


Figure 18. Interrupt Descriptor Table Definition

References to IDT entries are made via INT instructions, external interrupt vectors, or exceptions. The IDT must be at least 256 bytes in size to allocate space for all reserved interrupts.

Privilege

The 80286 has a four-level hierarchical privilege system which controls the use of privileged instructions and access to descriptors (and their associated segments) within a task. Four-level privilege, as shown in Figure 19, is an extension of the user/supervisor mode commonly found in minicomputers. The privilege levels are numbered 0 through 3. Level 0 is the most privileged level. Privilege

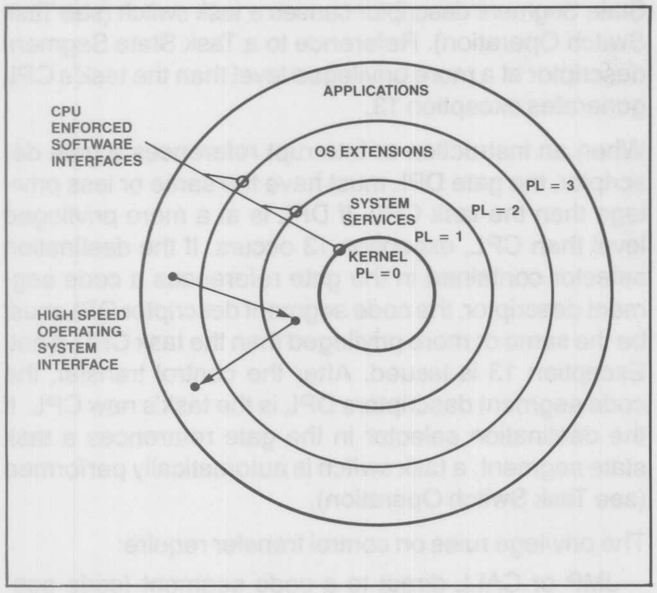


Figure 19. Hierarchical Privilege Levels

levels provide protection within a task. (Tasks are isolated by providing private LDT's for each task.) Operating system routines, interrupt handlers, and other system software can be included and protected within the virtual address space of each task using the four levels of privilege. Tasks may also have a separate stack for each privilege level.

Tasks, descriptors, and selectors have a privilege level attribute that determines whether the descriptor may be used. Task privilege effects the use of instructions and descriptors. Descriptor and selector privilege only effect access to the descriptor.

TASK PRIVILEGE

A task always executes at one of the four privilege levels. The task privilege level at any specific instant is called the Current Privilege Level (CPL) and is defined by the lower two bits of the CS register. CPL cannot change during execution in a single code segment. A task's CPL may only be changed by control transfers through gate descriptors to a new code segment (See Control Transfer). Tasks begin executing at the CPL value specified by the code segment when the task is initiated via a task switch operation. A task executing at Level 0 can access all data segments defined in the GDT and the task's LDT and is considered the most trusted level. A task executing at Level 3 has the most restricted access to data and is considered the least trusted level.

DESCRIPTOR PRIVILEGE

Descriptor privilege is specified by the Descriptor Privilege Level (DPL) field of the descriptor access byte. DPL specifies the least trusted task privilege level (CPL) at which a task may access the descriptor. Descriptors with DPL = 0 are the most protected. Only tasks executing at privilege level 0 (CPL = 0) may access them. Descriptors with DPL = 3 are the least protected (i.e. have the least restricted access) since tasks can access them when CPL = 0, 1, 2, or 3. This rule applies to all descriptors, except LDT descriptors.

SELECTOR PRIVILEGE

Selector privilege is specified by the Requested Privilege Level (RPL) field in the least significant two bits of a selector. Selector RPL may establish a less trusted privilege level than the current privilege level for the use of a selector. This level is called the task's effective privilege level (EPL). RPL can only reduce the scope of a task's access to data with this selector. A task's effective privilege is the numeric maximum of RPL and CPL. A selector with RPL = 0 imposes no additional restriction on its use while a selector with RPL = 3 can only refer to segments at privilege Level 3 regardless of the task's CPL. RPL is generally used to verify that pointer parameters passed to a more trusted procedure are not allowed to use data at a more privileged level than the caller (refer to pointer testing instructions).

Descriptor Access and Privilege Validation

Determining the ability of a task to access a segment involves the type of segment to be accessed, the instruction used, the type of descriptor used and CPL, RPL, and DPL. The two basic types of segment accesses are control transfer (selectors loaded into CS) and data (selectors loaded into DS, ES or SS).

DATA SEGMENT ACCESS

Instructions that load selectors into DS and ES must refer to a data segment descriptor or readable code segment descriptor. The CPL of the task and the RPL of the selector must be the same as or more privileged (numerically equal to or lower than) than the descriptor DPL. In general, a task can only access data segments at the same or less privileged levels than the CPL or RPL (whichever is numerically higher) to prevent a program from accessing data it cannot be trusted to use.

An exception to the rule is a readable conforming code segment. This type of code segment can be read from any privilege level.

If the privilege checks fail (e.g. DPL is numerically less than the maximum of CPL and RPL) or an incorrect type of descriptor is referenced (e.g. gate descriptor or execute only code segment) exception 13 occurs. If the segment is not present, exception 11 is generated.

Instructions that load selectors into SS must refer to data segment descriptors for writable data segments. The descriptor privilege (DPL) and RPL must equal CPL. All other descriptor types or a privilege level violation will cause exception 13. A not present fault causes exception 12.

CONTROL TRANSFER

Four types of control transfer can occur when a selector is loaded into CS by a control transfer operation (see Table 10). Each transfer type can only occur if the operation which loaded the selector references the correct descriptor type. Any violation of these descriptor usage rules (e.g. JMP through a call gate or RET to a Task State Segment) will cause exception 13.

The ability to reference a descriptor for control transfer is also subject to rules of privilege. A CALL or JUMP instruction may only reference a code segment descriptor with DPL equal to the task CPL or a conforming segment with DPL of equal or greater privilege than CPL. The RPL of the selector used to reference the code descriptor must have as much privilege as CPL.

RET and IRET instructions may only reference code segment descriptors with descriptor privilege equal to or less privileged than the task CPL. The selector loaded into CS is the return address from the stack. After the return, the selector RPL is the task's new CPL. If CPL changes, the old stack pointer is popped after the return address.

When a JMP or CALL references a Task State Segment descriptor, the descriptor DPL must be the same or less privileged than the task's CPL. Reference to a valid Task State Segment descriptor causes a task switch (see Task Switch Operation). Reference to a Task State Segment descriptor at a more privileged level than the task's CPL generates exception 13.

When an instruction or interrupt references a gate descriptor, the gate DPL must have the same or less privilege than the task CPL. If DPL is at a more privileged level than CPL, exception 13 occurs. If the destination selector contained in the gate references a code segment descriptor, the code segment descriptor DPL must be the same or more privileged than the task CPL. If not, Exception 13 is issued. After the control transfer, the code segment descriptors DPL is the task's new CPL. If the destination selector in the gate references a task state segment, a task switch is automatically performed (see Task Switch Operation).

The privilege rules on control transfer require:

—JMP or CALL direct to a code segment (code segment descriptor) can only be to a conforming segment with DPL of equal or greater privilege than CPL or a non-conforming segment at the same privilege level.

—interrupts within the task or calls that may change privilege levels, can only transfer control through a gate at the same or a less privileged level than CPL to a code segment at the same or more privileged level than CPL.

—return instructions that don't switch tasks can only return control to a code segment at the same or less privileged level.

—task switch can be performed by a call, jump or interrupt which references either a task gate or task state segment at the same or less privileged level.

Table 10. Descriptor Access Rules for Control Transfer

Control Transfer Types	Operation Types	Descriptor Referenced	Descriptor Table
Intersegment within the same privilege level	JMP, CALL, RET, IRET*	Code Segment	GDT/LDT
Intersegment to the same or higher privilege level Interrupt within task may change CPL.	CALL	Call Gate	GDT/LDT
	Interrupt Instruction, Exception, External Interrupt	Trap or Interrupt Gate	IDT
Intersegment to a lower privilege level (changes task CPL)	RET, IRET*	Code Segment	GDT/LDT
Task Switch	CALL, JMP	Task State Segment	GDT
	CALL, JMP	Task Gate	GDT/LDT
	IRET** Interrupt Instruction, Exception, External Interrupt	Task Gate	IDT

*NT (Nested Task bit of flag word) = 0
**NT (Nested Task bit of flag word) = 1

PRIVILEGE LEVEL CHANGES

Any control transfer that changes CPL within the task, causes a change of stacks as part of the operation. Initial values of SS:SP for privilege levels 0, 1, and 2 are kept in the task state segment (refer to Task Switch Operation). During a JMP or CALL control transfer, the new stack pointer is loaded into the SS and SP registers and the previous stack pointer is pushed onto the new stack.

When returning to the original privilege level, its stack is restored as part of the RET or IRET instruction operation. For subroutine calls that pass parameters on the stack and cross privilege levels, a fixed number of words, as specified in the gate, are copied from the previous stack to the current stack. The inter-segment RET instruction with a stack adjustment value will correctly restore the previous stack pointer upon return.

Protection

The 80286 includes mechanisms to protect critical instructions that affect the CPU execution state (e.g. HLT) and code or data segments from improper usage. These mechanisms are grouped under the term "protection" and have three forms:

Restricted usage of segments (e.g. no write allowed to read-only data segments). The only segments available for use are defined by descriptors in the Local Descriptor Table (LDT) and Global Descriptor Table (GDT).

Restricted access to segments via the rules of privilege and descriptor usage.

Privileged instructions or operations that may only be executed at certain privilege levels as determined by the CPL and I/O Privilege Level (IOPL). The IOPL is defined by bits 14 and 13 of the flag word.

These checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

The IRET and POPF instructions do not perform some of their defined functions if CPL is not of sufficient privilege (numerically small enough). No exceptions or other indication are given when these conditions occur.

The IF bit is not changed if CPL > IOPL.

The IOPL field of the flag word is not changed if CPL > 0.

Table 11
Segment Register Load Checks

Error Description	Exception Number
Descriptor table limit exceeded	13
Segment descriptor not-present	11 or 12
Privilege rules violated	13
Invalid descriptor/segment type segment register load: —Read only data segment load to SS —Special control descriptor load to DS, ES, SS —Execute only segment load to DS, ES, SS —Data segment load to CS —Read/Execute code segment load to SS	13

Table 12. Operand Reference Checks

Error Description	Exception Number
Write into code segment	13
Read from execute-only code segment	13
Write to read-only data segment	13
Segment limit exceeded[1]	12 or 13

Note 1: Carry out in offset calculations is ignored.

Table 13. Privileged Instruction Checks

Error Description	Exception Number
CPL ≠ 0 when executing the following instructions: LIDT, LLDT, LGDT, LTR, LMSW, CTS, HLT	13
CPL > IOPL when executing the following instructions: INS, IN, OUTS, OUT, STI, CLI, LOCK	13

EXCEPTIONS

The 80286 detects several types of exceptions and interrupts, in protected mode (see Table 14). Most are restartable after the exceptional condition is removed. Interrupt handlers for most exceptions receive an error code, pushed on the stack after the return address, that identifies the selector involved (0 if none). The return address normally points to the failing instruction, including all leading prefixes. For a processor extension segment overrun exception, the return address will not point at the ESC instruction that caused the exception; however, the processor extension registers may contain the address of the failing instruction.

Table 14. Protected Mode Exceptions

Interrupt Vector	Function	Return Address At Failing Instruction?	Always Restart-able?	Error Code on Stack?
8	Double exception detected	Yes	No	Yes
9	Processor extension segment overrun	No	No	No
10	Invalid task state segment	Yes	Yes	Yes
11	Segment not present	Yes	Yes	Yes
12	Stack segment overrun or segment not present	Yes	Yes[1]	Yes
13	General protection	Yes	No	Yes

Note 1: When a PUSHA or POPA instruction attempts to wrap around the stack segment, the machine state after the exception will not be restartable. This condition is identified by the value of the saved SP being either 0000(H), 0001(H), FFFE(H), or FFFF(H).

All these checks are performed for all instructions and can be split into three categories: segment load checks (Table 11), operand reference checks (Table 12), and privileged instruction checks (Table 13). Any violation of the rules shown will result in an exception. A not-present exception related to the stack segment causes exception 12.

Special Operations

TASK SWITCH OPERATION

The 80286 provides a built-in task switch operation which saves the entire 80286 execution state (registers, address space, and a link to the previous task), loads a new execution state, and commences execution in the new task. Like gates, the task switch operation is invoked by executing an inter-segment JMP or CALL instruction which refers to a Task State Segment (TSS) or task gate descriptor in the GDT or LDT. An INT n instruction, exception, or external interrupt may also invoke the task switch operation by selecting a task gate descriptor in the associated IDT descriptor entry.

The TSS descriptor points at a segment (see Figure 20) containing the entire 80286 execution state while a task gate descriptor contains a TSS selector. The limit field must be > 002B(H).

Each task must have a TSS associated with it. The current TSS is identified by a special register in the 80286 called the Task Register (TR). This register contains a selector referring to the task state segment descriptor that defines the current TSS. A hidden base and limit register associated with TR are loaded whenever TR is loaded with a new selector.

The IRET instruction is used to return control to the task that called the current task or was interrupted. Bit 14 in the flag register is called the Nested Task (NT) bit. It controls the function of the IRET instruction. If NT = 0, the IRET instruction performs the regular current task return; when NT = 1, IRET performs a task switch operation back to the previous task.

When a CALL or INT instruction initiates a task switch, the old and new TSS will be marked busy and the back link field of the new TSS set to the old TSS selector. The NT bit of the new task is set by CALL or INT initiated task switches. An interrupt that does not cause a task switch will clear NT. NT may also be set or cleared by POPF or IRET instructions.

The task state segment is marked busy by changing the descriptor type field from Type 1 to Type 3. Use of a selector that references a busy task state segment causes Exception 13.

PROCESSOR EXTENSION CONTEXT SWITCHING

The context of a processor extension (such as the 80287 numerics processor) is not changed by the task switch operation. A processor extension context need only be changed when a different task attempts to use the processor extension (which still contains the context of a previous task). The 80286 detects the first use of a processor extension after a task switch by causing the processor extension not present exception (7). The interrupt handler may then decide whether a context change is necessary.

Whenever the 80286 switches tasks, it sets the Task Switched (TS) bit of the MSW. TS indicates that a processor extension context may belong to a different task than the current one. The processor extension not present exception (7) will occur when attempting to execute an ESC or WAIT instruction if TS = 1 and a processor extension is present (MP = 1 in MSW).

POINTER TESTING INSTRUCTIONS

The iAPX 80286 provides several instructions to speed pointer testing and consistency checks for maintaining system integrity (see Table 15). These instructions use the memory management hardware to verify that a selector value refers to an appropriate segment without risking an exception. A condition flag indicates whether use of the selector or segment will cause an exception.

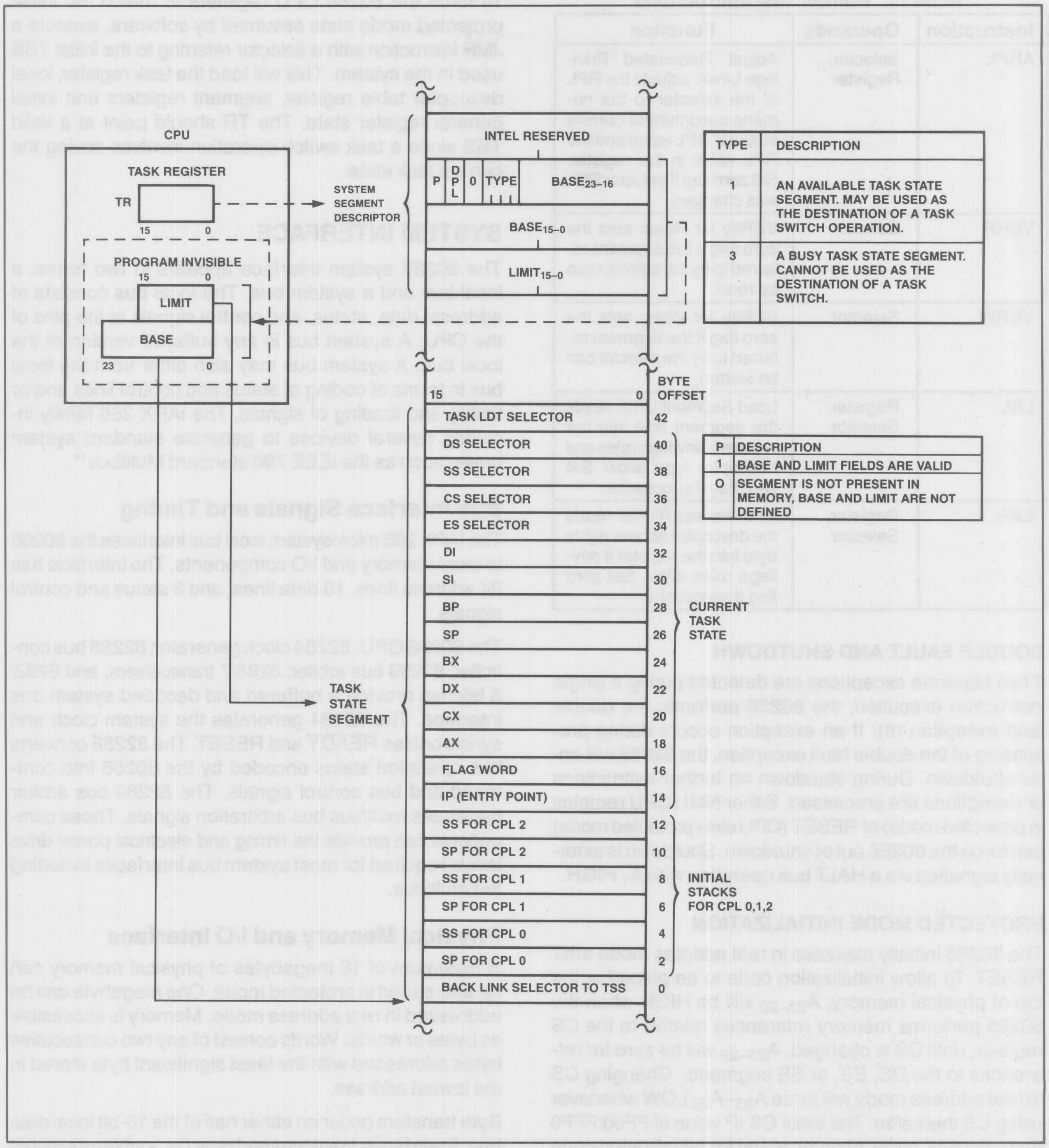


Figure 20. Task State Segment and TSS Registers

AFN-02060A

Table 15. Pointer Test Instructions

Instruction	Operands	Function
ARPL	Selector, Register	Adjust Requested Privilege Level: adjusts the RPL of the selector to the numeric maximum of current selector RPL value and the RPL value in the register. Set zero flag if selector RPL was changed.
VERR	Selector	VERify for Read: sets the zero flag if the segment referred to by the selector can be read.
VERW	Selector	VERify for Write: sets the zero flag if the segment referred to by the selector can be written.
LSL	Register, Selector	Load Segment Limit: reads the segment limit into the register if privilege rules and descriptor type allow. Set zero flag if successful.
LAR	Register, Selector	Load Access Rights: reads the descriptor access rights byte into the register if privilege rules allow. Set zero flag if successful.

DOUBLE FAULT AND SHUTDOWN

If two separate exceptions are detected during a single instruction execution, the 80286 performs the double fault exception (8). If an exception occurs during processing of the double fault exception, the 82086 will enter shutdown. During shutdown no further instructions or exceptions are processed. Either NMI (CPU remains in protected mode) or RESET (CPU exits protected mode) can force the 80286 out of shutdown. Shutdown is externally signalled via a HALT bus operation with A_1 HIGH.

PROTECTED MODE INITIALIZATION

The 80286 initially executes in real address mode after RESET. To allow initialization code to be placed at the top of physical memory, A_{23-20} will be HIGH when the 80286 performs memory references relative to the CS register, until CS is changed. A_{23-20} will be zero for references to the DS, ES, or SS segments. Changing CS in real address mode will force A_{23}–A_{20} LOW whenever using CS thereafter. The initial CS:IP value of FF00:FFF0 provides 64K bytes of code space for initialization code without changing CS.

Before placing the 80286 into protected mode, several registers must be initialized. The GDT and IDT base registers must refer to a valid GDT and IDT. After executing the LMSW instruction to set PE, the 80286 must immediately execute an intra-segment JMP instruction to clear the instruction queue of instructions decoded in real address mode.

To force the 80286 CPU registers to match the initial protected mode state assumed by software, execute a JMP instruction with a selector referring to the initial TSS used in the system. This will load the task register, local descriptor table register, segment registers and initial general register state. The TR should point at a valid TSS since a task switch operation involves saving the current task state.

SYSTEM INTERFACE

The 80286 system interface appears in two forms: a local bus and a system bus. The local bus consists of address, data, status, and control signals at the pins of the CPU. A system bus is any buffered version of the local bus. A system bus may also differ from the local bus in terms of coding of status and control lines and/or timing and loading of signals. The iAPX 286 family includes several devices to generate standard system buses such as the IEEE 796 standard Multibus™.

Bus Interface Signals and Timing

The iAPX 286 microsystem local bus interfaces the 80286 to local memory and I/O components. The interface has 24 address lines, 16 data lines, and 8 status and control signals.

The 80286 CPU, 82284 clock generator, 82288 bus controller, 82289 bus arbiter, 8286/7 transceivers, and 8282/3 latches provide a buffered and decoded system bus interface. The 82284 generates the system clock and synchronizes $\overline{\text{READY}}$ and RESET. The 82288 converts bus operation status encoded by the 80286 into command and bus control signals. The 82289 bus arbiter generates multibus bus arbitration signals. These components can provide the timing and electrical power drive levels required for most system bus interfaces including the multibus.

Physical Memory and I/O Interface

A maximum of 16 megabytes of physical memory can be addressed in protected mode. One megabyte can be addressed in real address mode. Memory is accessible as bytes or words. Words consist of any two consecutive bytes addressed with the least significant byte stored in the lowest address.

Byte transfers occur on either half of the 16-bit local data bus. Even bytes are accessed over D_{7-0} while odd bytes are transferred over D_{15-8}. Even-addressed words are transferred over D_{15-0} in one bus cycle, while odd-addressed words require two bus operations. The first transfers data on D_{15-8}, and the second transfers data on D_{7-0}. Both byte data transfers occur automatically, transparent to software.

Two bus signals, A_0 and $\overline{\text{BHE}}$, control transfers over the lower and upper halves of the data bus. Even address byte transfers are indicated by A_0 LOW and $\overline{BHE}$ HIGH. Odd address byte transfers are indicated by A_0 HIGH and $\overline{BHE}$ LOW. Both A_0 and $\overline{BHE}$ are LOW for even address word transfers.

The I/O address space contains 64K addresses in both modes. The I/O space is accessible as either bytes or words, as is memory. Byte wide peripheral devices may be attached to either the upper or lower byte of the data bus. Byte-wide I/O devices attached to the upper data byte (D_{15-8}) are accessed with odd I/O addresses. Devices on the lower data byte are accessed with even I/O addresses. An interrupt controller such as Intel's 8259A must be connected to the lower data byte (D_{7-0}) for proper return of the interrupt vector.

Bus Operation

The 80286 uses a double frequency system clock (CLK input) to control bus timing. All signals on the local bus are measured relative to the system CLK input. The CPU divides the system clock by 2 to produce the internal processor clock, which determines bus state. Each processor clock is composed of two system clock cycles named phase 1 and phase 2. The 82284 clock generator output (PCLK) identifies the next phase of the processor clock. (See Figure 21.)

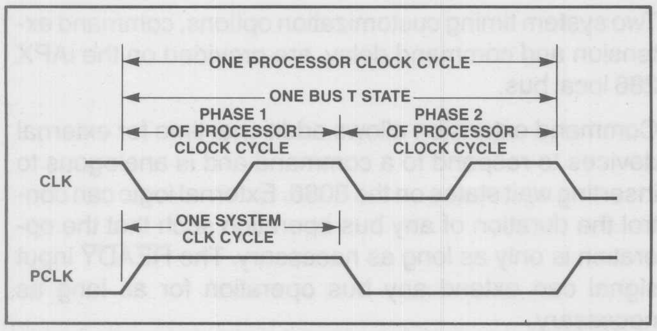

ONE PROCESSOR CLOCK CYCLE
ONE BUS T STATE
PHASE 1 OF PROCESSOR CLOCK CYCLE
PHASE 2 OF PROCESSOR CLOCK CYCLE
CLK
ONE SYSTEM CLK CYCLE
PCLK

Figure 21. System and Processor Clock Relationships

Six types of bus operations are supported; memory read, memory write, I/O read, I/O write, interrupt acknowledge, and halt/shutdown. Data can be transferred at a maximum rate of one word per two processor clock cycles.

The iAPX 286 bus has three basic states: idle (T_i), send status (T_s), and perform command (T_c). The 80286 CPU also has a fourth local bus state called hold (T_h). T_h indicates that the 80286 has surrendered control of the local bus to another bus master in response to a HOLD request.

Each bus state is one processor clock long. Figure 22 shows the four 80286 local bus states and allowed transitions.

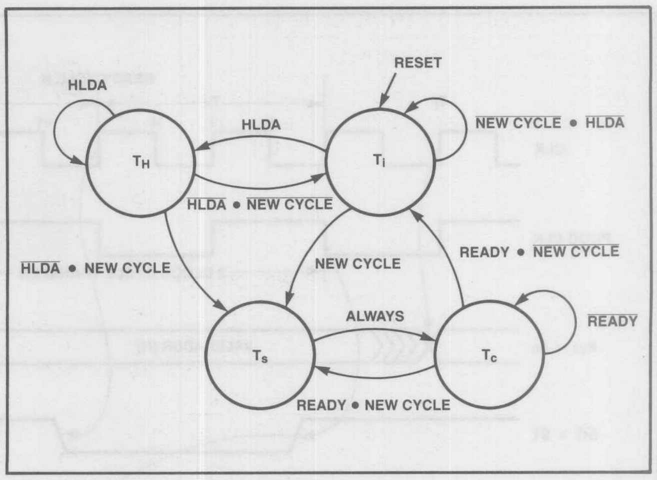

RESET
HLDA
HLDA
NEW CYCLE • $\overline{HLDA}$
T_H
T_i
$\overline{HLDA}$ • NEW CYCLE
READY • $\overline{NEW CYCLE}$
$\overline{HLDA}$ • NEW CYCLE
NEW CYCLE
ALWAYS
$\overline{READY}$
T_s
T_c
READY • NEW CYCLE

Figure 22. 80286 Bus States

Bus States

The idle (T_i) state indicates that no data transfers are in progress or requested. The first active state, T_s is signalled by either status line $\overline{S1}$ or $\overline{S0}$ going LOW also identifying phase 1 of the processor clock. During T_s, the command encoding, the address, and data (for a write operation) are available on the 80286 output pins. The 82288 bus controller decodes the status signals and generates Multibus compatible read/write command and local transceiver control signals.

After T_s, the perform command (T_c) state is entered. Memory or I/O devices respond to the bus operation during T_c, either transferring read data to the CPU or accepting write data. T_c states may be repeated as often as necessary to assure sufficient time for the memory or I/O device to respond. The $\overline{READY}$ signal determines whether T_c is repeated.

During hold (T_h), the 80286 will float all address, data, and status output pins enabling another bus master to use the local bus. The 80286 HOLD input signal is used to place the 80286 into the T_h state. The 80286 HLDA output signal indicates that the CPU has entered T_h.

Pipelined Addressing

The 80286 uses a local bus interface with pipelined timing to allow as much time as possible for data access. Pipelined timing allows bus operations to be performed in two processor cycles, while allowing each individual bus operation to last for three processor cycles.

The timing of the address outputs is pipelined such that the address of the next bus operation becomes available during the current bus operation. Or in other words, the first clock of the next bus operation is overlapped with the last clock of the current bus operation. Therefore, address decode and routing logic can operate in ad-

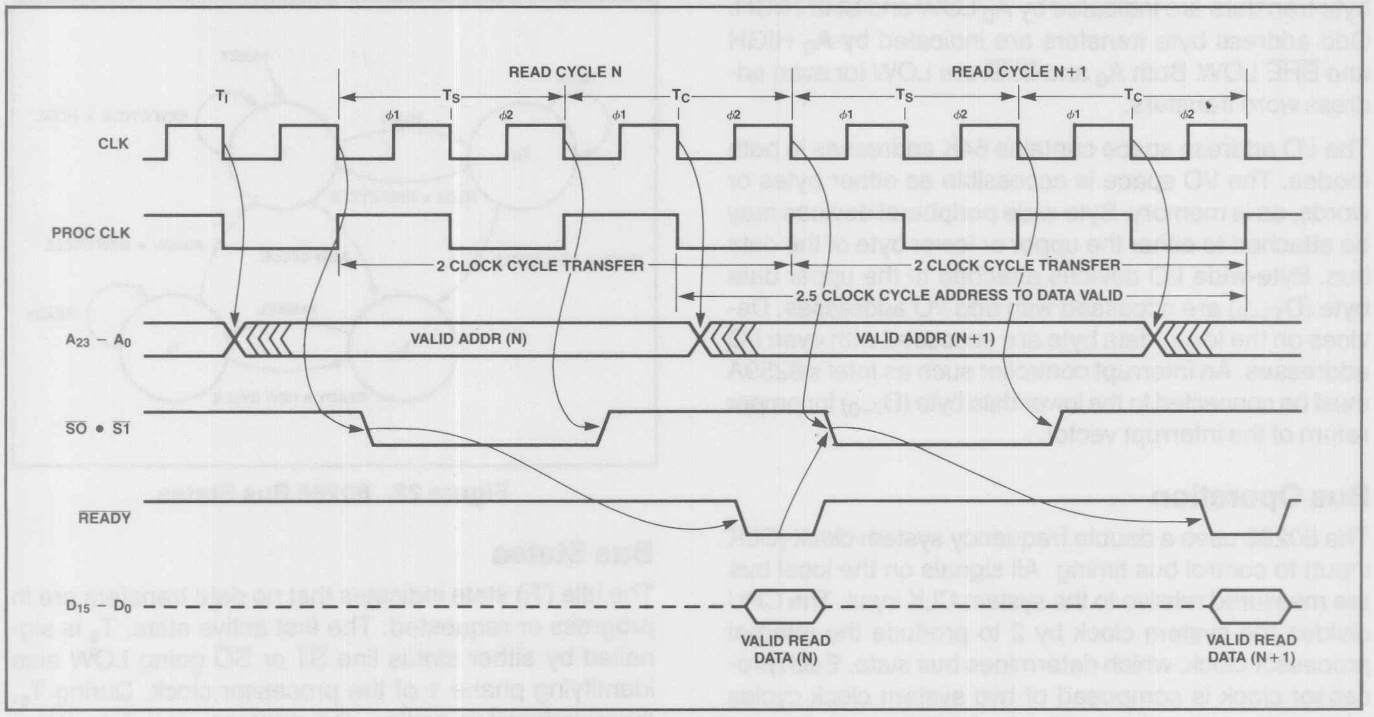


Figure 23. Basic Bus Cycle

vance of the next bus operation. External address latches may hold the address stable for the entire bus operation, and provide additional AC and DC buffering.

The 80286 does not maintain the address of the current bus operation during all T_C states. Instead, the address for the next bus operation may be emitted during phase 2 of any T_C. The address remains valid during phase 1 of the first T_C to guarantee hold time, relative to ALE, for the address latch inputs.

Bus Control Signals

The 82288 bus controller provides control signals; address latch enable (ALE), Read/Write commands, data transmit/receive (DT/R̄), and data enable (DEN) that control the address latches, data transceivers, write enable, and output enable for memory and I/O systems.

The Address Latch Enable (ALE) output determines when the address may be latched. ALE provides at least one system CLK period of address hold time from the end of the previous bus operation until the address for the next bus operation appears at the latch outputs. This address hold time is required to support Multibus® and common memory systems.

The data bus transceivers are controlled by 82288 outputs Data Enable (DEN) and Data Transmit/Receive (DT/R̄). DEN enables the data transceivers; while DT/R̄ controls transceiver direction. DEN and DT/R̄ are timed to prevent bus contention between the bus master, data bus transceivers, and system data bus tranceivers.

Command Timing Controls

Two system timing customization options, command extension and command delay, are provided on the iAPX 286 local bus.

Command extension allows additional time for external devices to respond to a command and is analogous to inserting wait states on the 8086. External logic can control the duration of any bus operation such that the operation is only as long as necessary. The READY input signal can extend any bus operation for as long as necessary.

Command delay allows an increase of address or write data setup time to system bus command active for any bus operation by delaying when the system bus command becomes active. Command delay is controlled by the 82288 CMDLY input. After T_S, the bus controller samples CMDLY at each failing edge of CLK. If CMDLY is HIGH, the 82288 will not activate the command signal. When CMDLY is LOW, the 82288 will activate the command signal. After the command becomes active, the CMDLY input is not sampled.

When a command is delayed, the available response time from command active to return read data or accept write data is less. To customize system bus timing, an address decoder can determine which bus operations require delaying the command. The CMDLY input does not affect the timing of ALE, DEN, or DT/R̄.

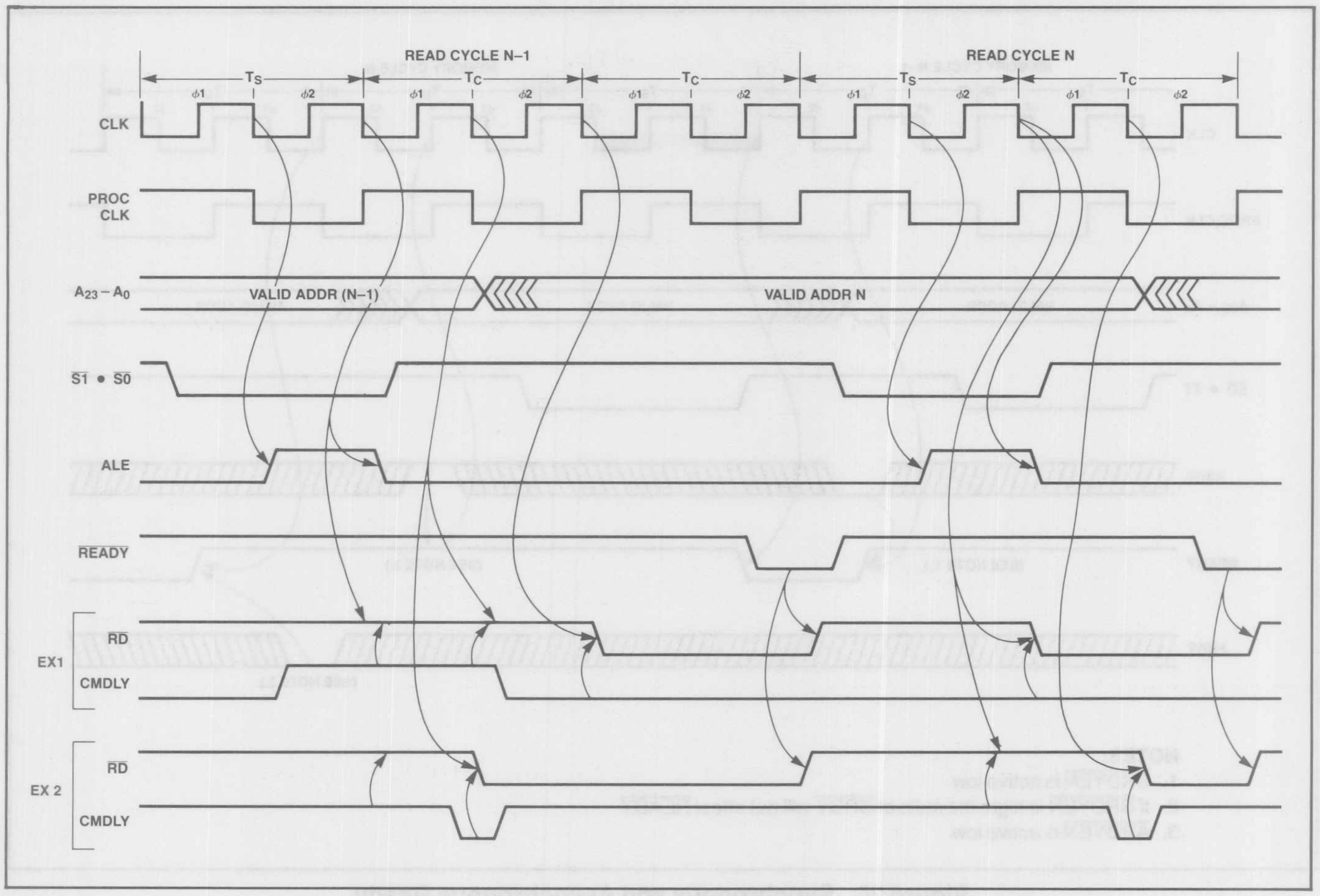


Figure 24. CMDLY Controls and Leading Edge of the Command

Figure 24 illustrates four uses of CMDLY. Example 1 shows delaying the read command two system CLKs for cycle N-1 and no delay for cycle N, and example 2 shows delaying the read command one system CLK for cycle N-1 and one system CLK delay for cycle N.

Bus Cycle Termination

At maximum transfer rates, the iAPX 286 bus alternates between the status and command states. The bus status signals become inactive after T_S so that they may correctly signal the start of the next bus operation after the completion of the current cycle. No external indication of T_C exists on the iAPX 286 local bus. The bus master and bus controller enter T_C directly after T_S and continue executing T_C cycles until terminated by $\overline{\text{READY}}$.

$\overline{\text{READY}}$ Operation

The current bus master and 82288 bus controller terminate each bus operation simultaneously to achieve maximum bus bandwidth. Both are informed in advance by $\overline{\text{READY}}$ active which identifies the last T_C cycle of the current bus operation. The bus master and bus controller must see the same sense of the $\overline{\text{READY}}$ signal, thereby requiring $\overline{\text{READY}}$ be synchronous to the system clock.

Synchronous Ready

The 82284 clock generator provides $\overline{\text{READY}}$ synchronization from both synchronous and asynchronous sources (see Figure 25). The synchronous ready input ($\overline{\text{SRDY}}$) of the clock generator is sampled with the falling edge of CLK at the end of phase 1 of each T_C. The state of $\overline{\text{SRDY}}$ is then broadcast to the bus master and bus controller via the $\overline{\text{READY}}$ output line.

Asynchronous Ready

Many systems have devices or subsystems that are asynchronous to the system clock. As a result, their ready outputs cannot be guaranteed to meet the 82284 $\overline{\text{SRDY}}$ setup and hold time requirements. The 82284 asynchronous ready input ($\overline{\text{ARDY}}$) is designed to accept such signals. The $\overline{\text{ARDY}}$ input is sampled at the beginning of each T_C cycle by 82284 synchronization logic. This provides a system CLK cycle time to resolve its value before broadcasting it to the bus master and bus controller.

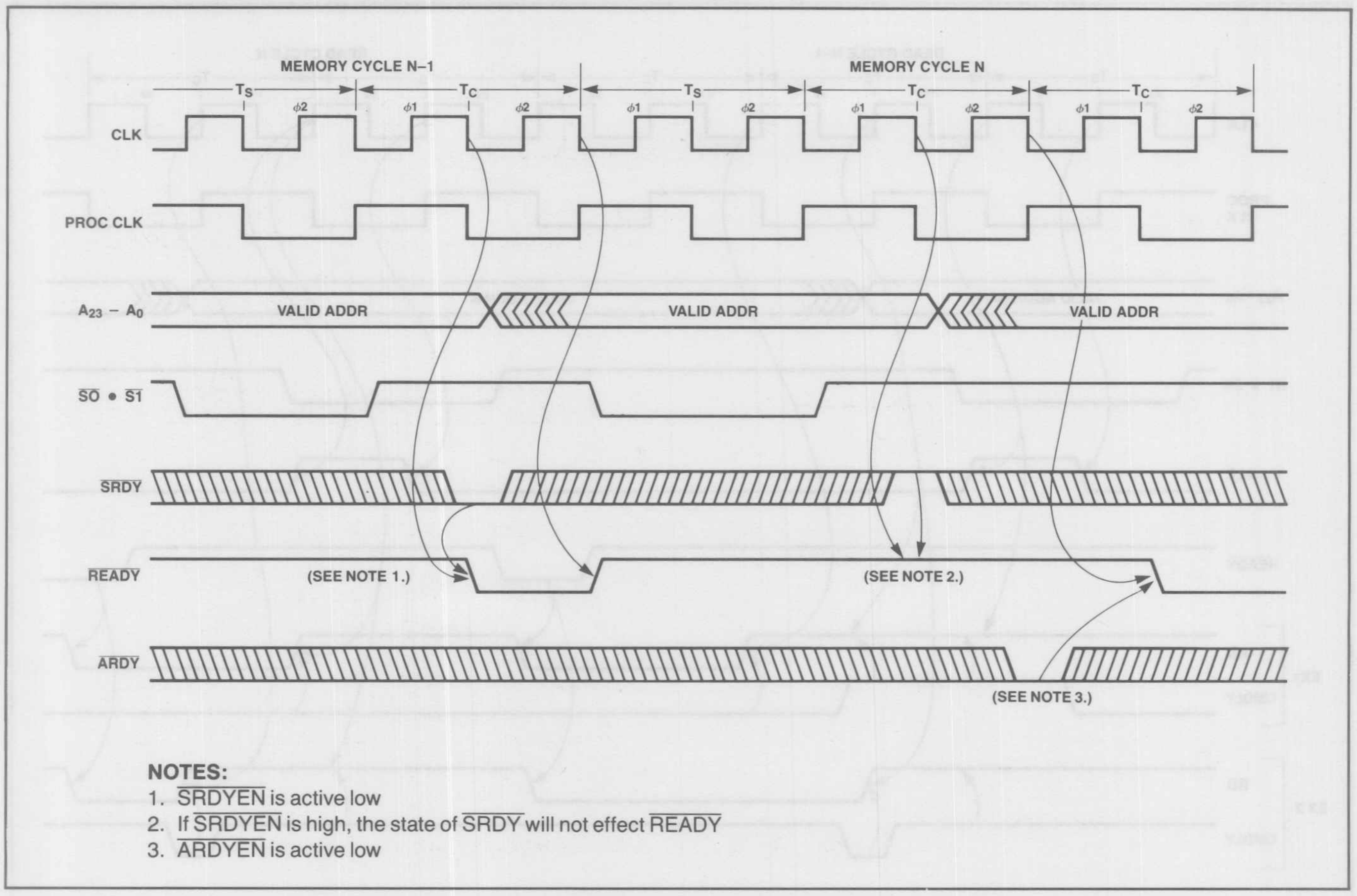


Figure 25. Synchronous and Asynchronous Ready

Each ready input of the 82284 has an enable pin (SRDYEN and ARDYEN) to select whether the current bus operation will be terminated by the synchronous or asynchronous ready. Either of the ready inputs may terminate a bus operation. These enable inputs are active low and have the same timing as their respective ready inputs. Address decode logic usually selects whether the current bus operation should be terminated by ARDY or SRDY.

Data Bus Control

Figures 26, 27, and 28 show how the DT/R, DEN, data bus, and address signals operate for different combinations of read, write, and idle bus operations. DT/R goes active (LOW) for a read operaton. DT/R remains HIGH before, during, and between write operations.

The data bus is driven with write data during the second phase of T_S. The delay in write data timing allows the read data drivers, from a previous read cycle, sufficient time to enter 3-state OFF before the 80286 CPU begins driving the local data bus for write operations. Write data will always remain valid for one system clock past the last T_C to provide sufficient hold time for Multibus or other similar memory or I/O systems. During write-read or write-idle sequences the data bus enters 3-state OFF during the second phase of the processor cycle after the last T_C. In a write-write sequence the data bus does not enter 3-state OFF between T_C and T_S.

Bus Usage

The 80286 local bus may be used for several functions: instruction data transfers, data transfers by other bus masters, instruction fetching, processor extension data transfers, interrupt acknowledge, and halt/shutdown. This section describes local bus activities which have special signals or requirements.

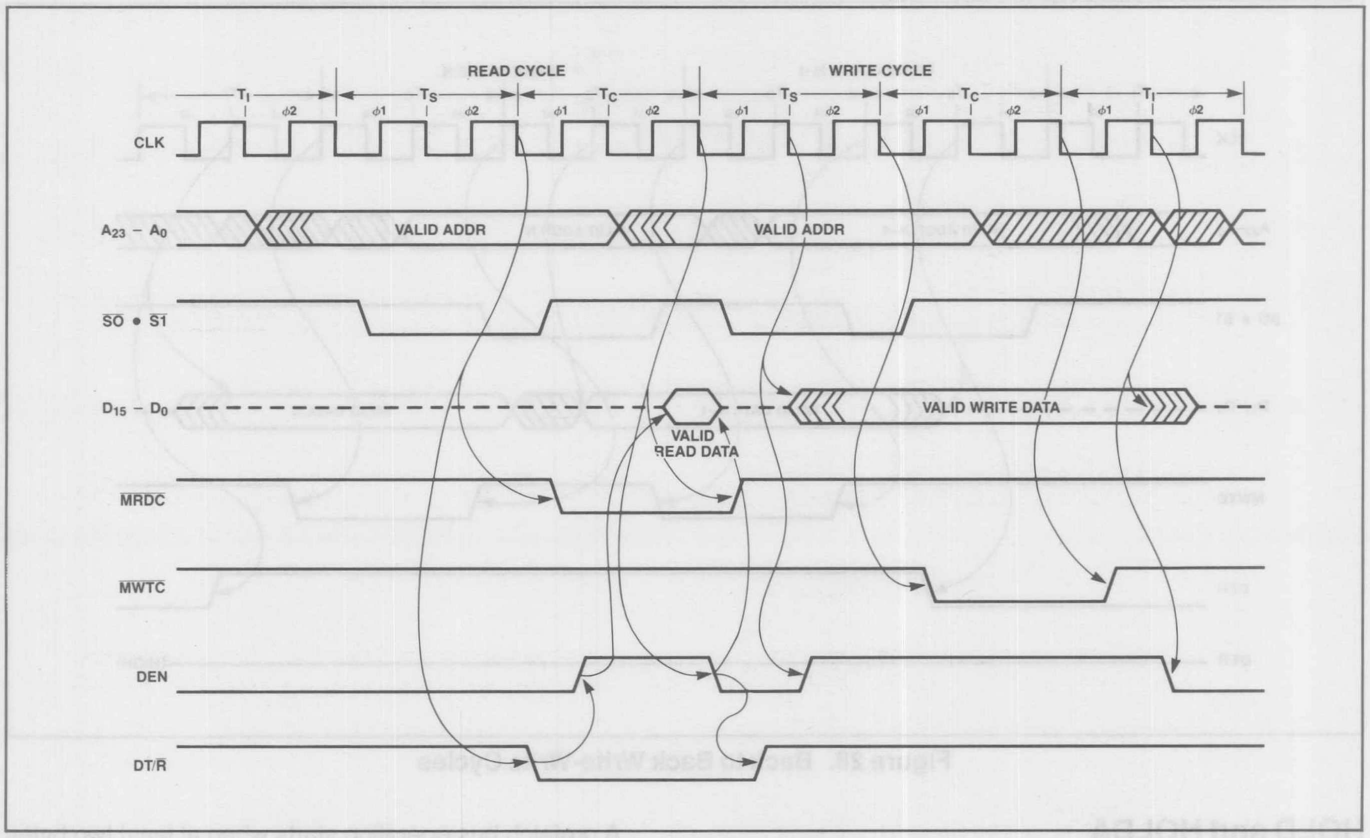


Figure 26. Back to Back Read-Write Cycles

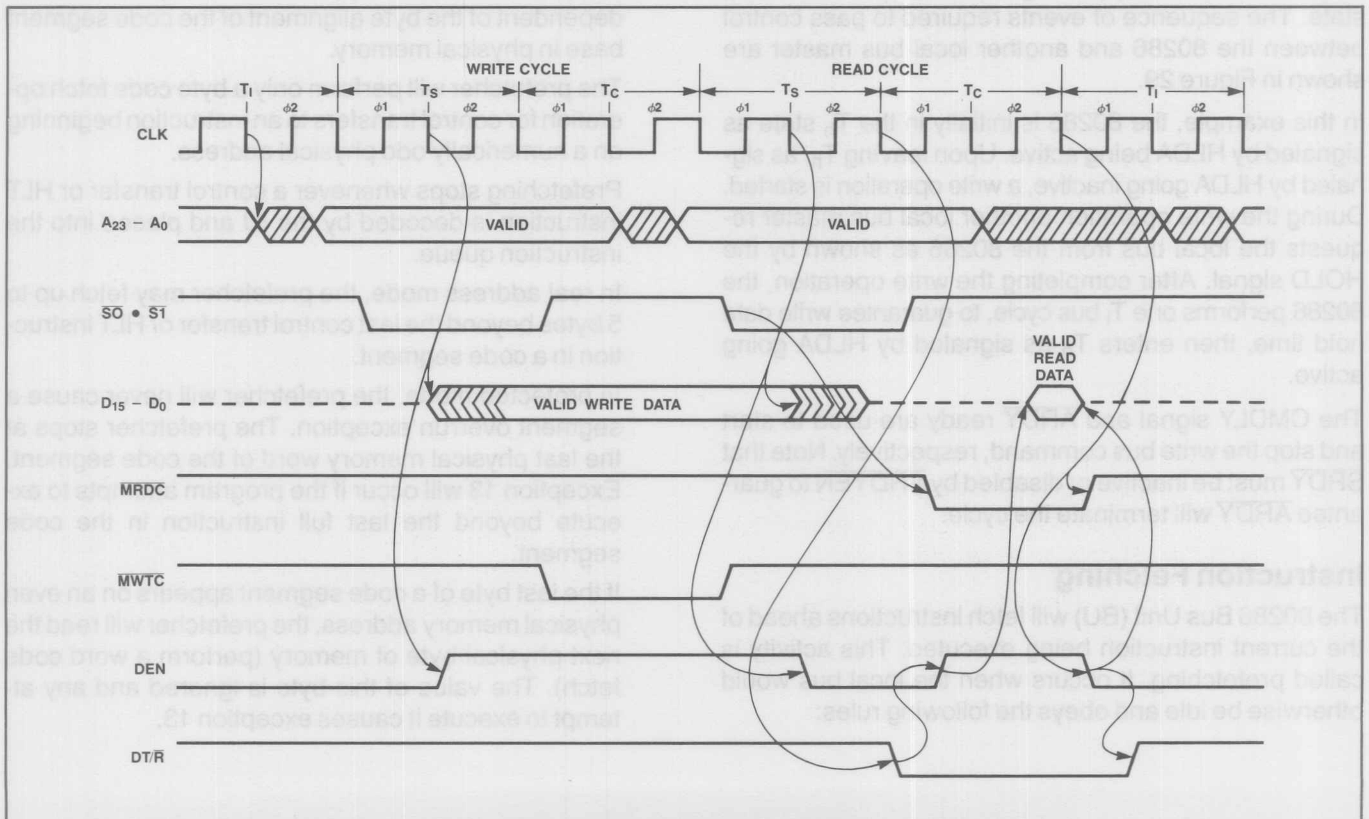


Figure 27. Back to Back Write-Read Cycles

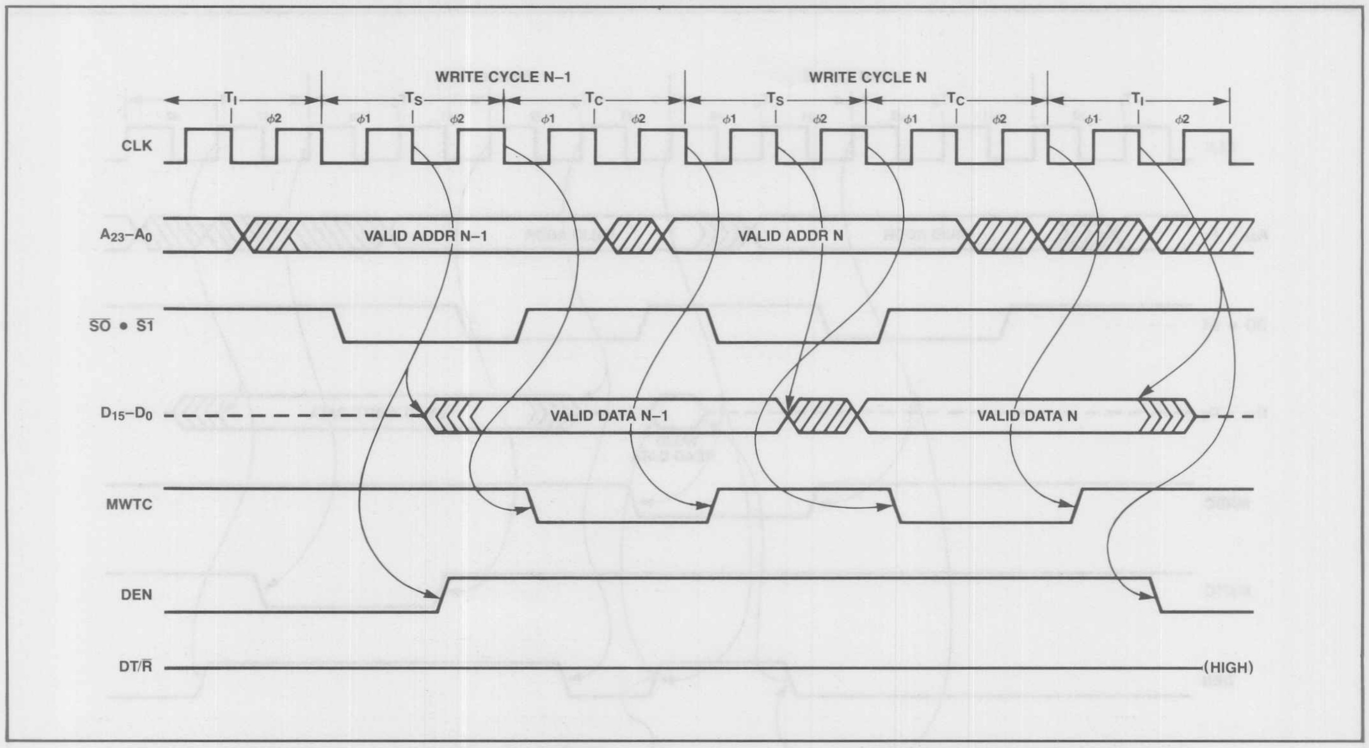


Figure 28. Back to Back Write-Write Cycles

HOLD and HOLDA

HOLD and HLDA allow another bus master to gain control of the local bus by placing the 80286 bus into the T_h state. The sequence of events required to pass control between the 80286 and another local bus master are shown in Figure 29.

In this example, the 80286 is initially in the T_h state as signaled by HLDA being active. Upon leaving T_h, as signaled by HLDA going inactive, a write operation is started. During the write operation another local bus master requests the local bus from the 80286 as shown by the HOLD signal. After completing the write operation, the 80286 performs one T_i bus cycle, to guarantee write data hold time, then enters T_h as signaled by HLDA going active.

The CMDLY signal and $\overline{\text{ARDY}}$ ready are used to start and stop the write bus command, respectively. Note that $\overline{\text{SRDY}}$ must be inactive or disabled by $\overline{\text{SRDYEN}}$ to guarantee $\overline{\text{ARDY}}$ will terminate the cycle.

Instruction Fetching

The 80286 Bus Unit (BU) will fetch instructions ahead of the current instruction being executed. This activity is called prefetching. It occurs when the local bus would otherwise be idle and obeys the following rules:

A prefetch bus operation starts when at least two bytes of the 6-byte prefetch queue are empty.

The prefetcher normally performs word prefetches independent of the byte alignment of the code segment base in physical memory.

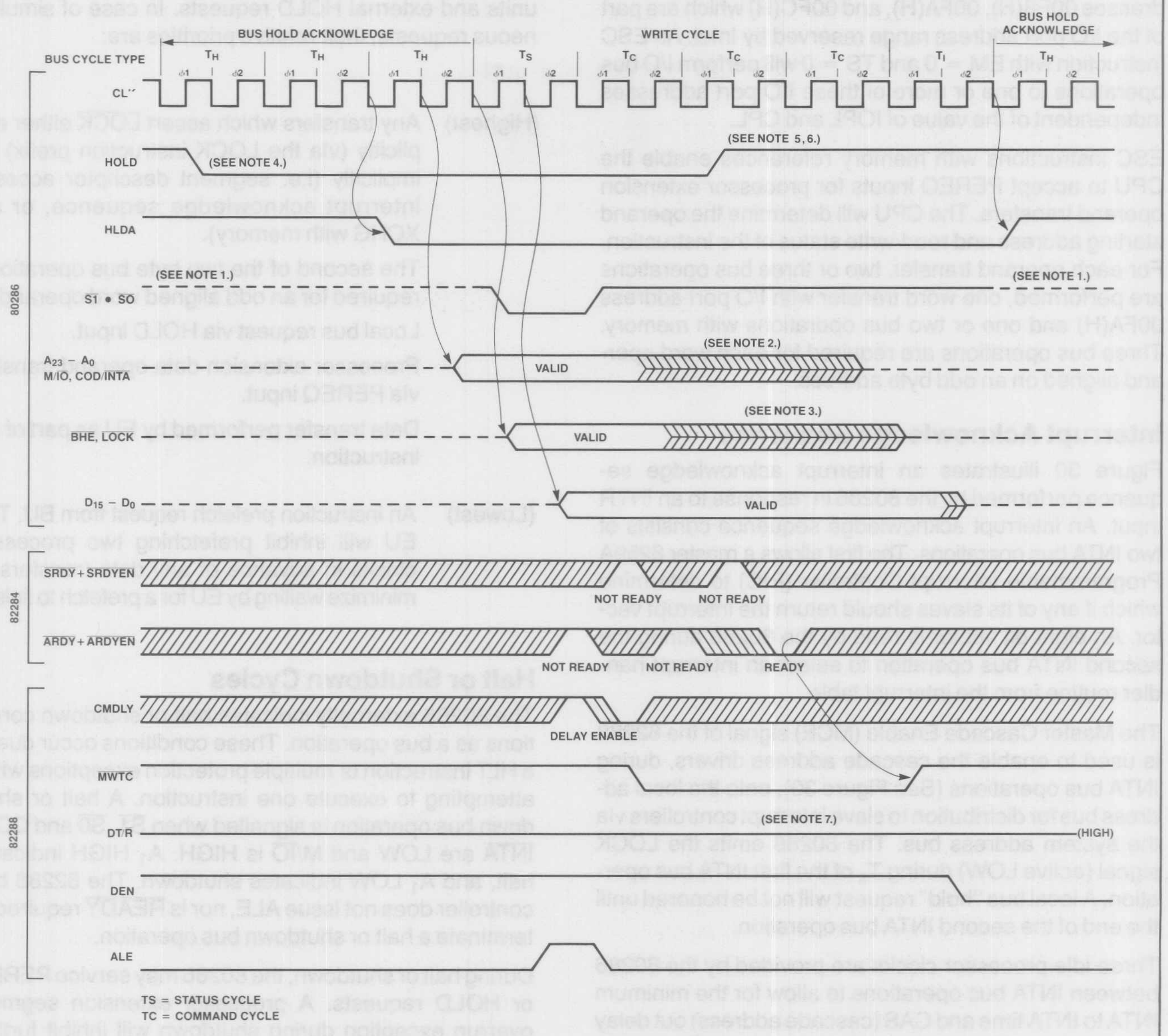
The prefetcher will perform only a byte code fetch operation for control transfers to an instruction beginning on a numerically odd physical address.

Prefetching stops whenever a control transfer or HLT instruction is decoded by the IU and placed into the instruction queue.

In real address mode, the prefetcher may fetch up to 5 bytes beyond the last control transfer or HLT instruction in a code segment.

In protected mode, the prefetcher will never cause a segment overrun exception. The prefetcher stops at the last physical memory word of the code segment. Exception 13 will occur if the program attempts to execute beyond the last full instruction in the code segment.

If the last byte of a code segment appears on an even physical memory address, the prefetcher will read the next physical byte of memory (perform a word code fetch). The value of this byte is ignored and any attempt to execute it causes exception 13.



NOTES:
1. Status lines are not driven by 80286, yet remain high due to pullup resistors in 82288 and 82289 during HOLD state.
2. Address, M/$\overline{\text{IO}}$ and COD/$\overline{\text{INTA}}$ may start floating during any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD. The float starts in $\phi 2$ of TC.
3. $\overline{\text{BHE}}$ and $\overline{\text{LOCK}}$ may start floating after the end of any TC depending on when internal 80286 bus arbiter decides to release bus to external HOLD.
4. The minimum HOLD ↓ to HLDA ↓ time is shown. Maximus is one T_H longer.
5. The earliest HOLD ↑ time is shown which will always allow a subsequent memory cycle if pending.
6. The minimum HOLD ↑ to HLDA ↑ time is shown. Maximum is a function of the instruction, type of bus cycle and other machine status (i.e., Interrupts, Waits, Lock, etc.)
7. Asynchronous ready allows termination of the cycle. Synchronous ready does not signal ready in this example. Synchronous ready state is ignored after ready is signaled via the asynchronous input.

Figure 29. Multibus Write Terminated by Asynchronous Ready

Processor Extension Transfers

The processor extension interface uses I/O port addresses 00F8(H), 00FA(H), and 00FC(H) which are part of the I/O port address range reserved by Intel. An ESC instruction with EM = 0 and TS = 0 will perform I/O bus operations to one or more of these I/O port addresses independent of the value of IOPL and CPL.

ESC instructions with memory references enable the CPU to accept PEREQ inputs for processor extension operand transfers. The CPU will determine the operand starting address and read/write status of the instruction. For each operand transfer, two or three bus operations are performed, one word transfer with I/O port address 00FA(H) and one or two bus operations with memory. Three bus operations are required for each word operand aligned on an odd byte address.

Interrupt Acknowledge Sequence

Figure 30 illustrates an interrupt acknowledge sequence performed by the 80286 in response to an INTR input. An interrupt acknowledge sequence consists of two INTA bus operations. The first allows a master 8259A Programmable Interrupt Controller (PIC) to determine which if any of its slaves should return the interrupt vector. An eight bit vector is read by the 80286 during the second INTA bus operation to select an interrupt handler routine from the interrupt table.

The Master Cascade Enable (MCE) signal of the 82288 is used to enable the cascade address drivers, during INTA bus operations (See Figure 30), onto the local address bus for distribution to slave interrupt controllers via the system address bus. The 80286 emits the $\overline{\text{LOCK}}$ signal (active LOW) during T_S of the first INTA bus operation. A local bus "hold" request will not be honored until the end of the second INTA bus operation.

Three idle processor clocks are provided by the 80286 between INTA bus operations to allow for the minimum INTA to INTA time and CAS (cascade address) out delay of the 8259A. The second INTA bus operation must always have at least one extra T_C state added via logic controlling $\overline{\text{READY}}$. A_{23}–A_0 are in 3-state OFF until after the first T_C state of the second INTA bus operation. This prevents bus contention between the cascade address drivers and CPU address drivers. The extra T_C state allows time for the 80286 to resume driving the address lines for subsequent bus operations.

Local Bus Usage Priorities

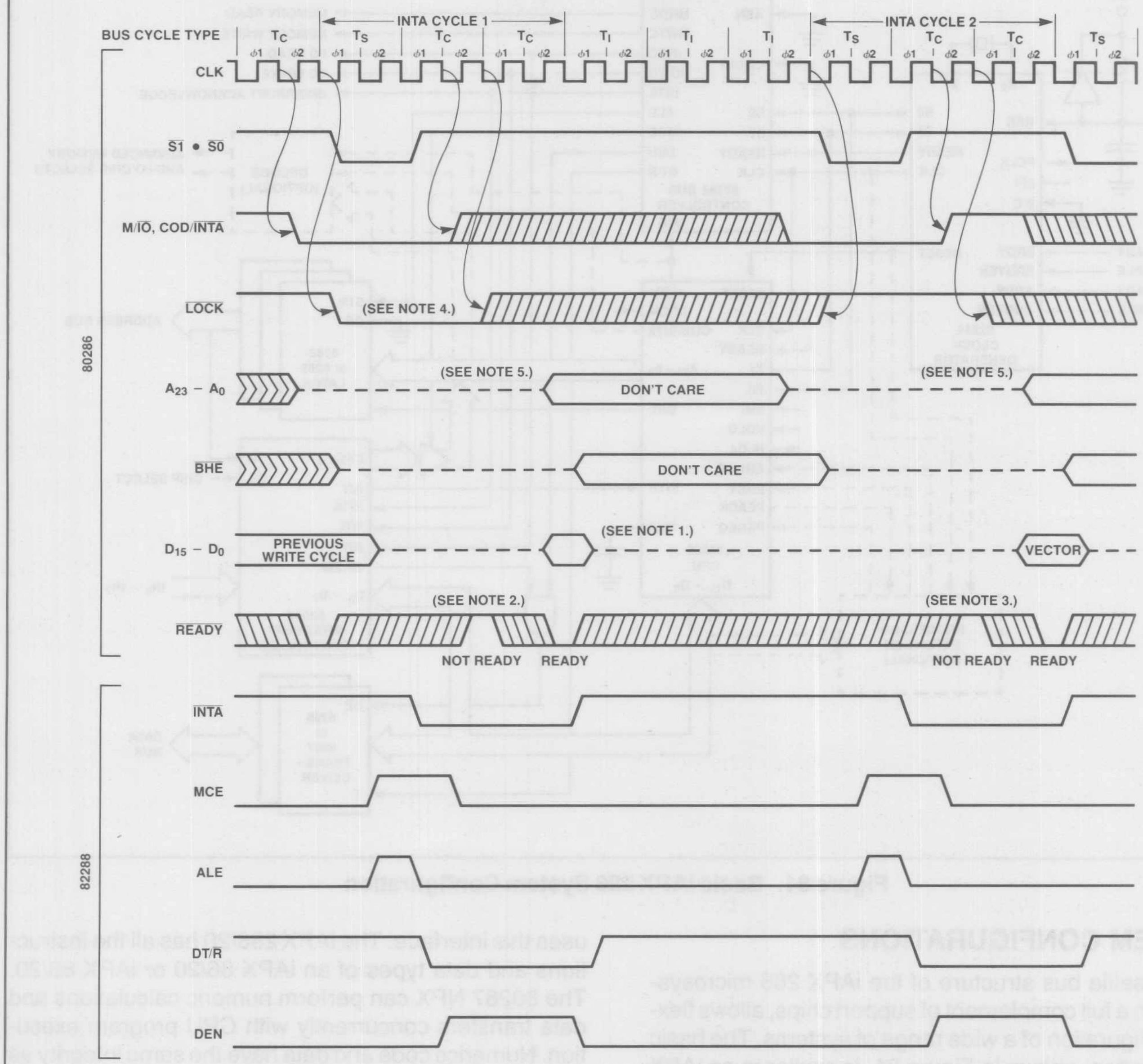
The 80286 local bus is shared among several internal units and external HOLD requests. In case of simultaneous requests, their relative priorities are:

(Highest)	Any transfers which assert $\overline{\text{LOCK}}$ either explicitly (via the LOCK instruction prefix) or implicitly (i.e. segment descriptor access, interrupt acknowledge sequence, or an XCHG with memory).
	The second of the two byte bus operations required for an odd aligned word operand.
	Local bus request via HOLD input.
	Processor extension data operand transfer via PEREQ input.
	Data transfer performed by EU as part of an instruction.
(Lowest)	An instruction prefetch request from BU. The EU will inhibit prefetching two processor clocks in advance of any data transfers to minimize waiting by EU for a prefetch to finish.

Halt or Shutdown Cycles

The 80286 externally indicates halt or shutdown conditions as a bus operation. These conditions occur due to a HLT instruction or multiple protection exceptions while attempting to execute one instruction. A halt or shutdown bus operation is signalled when $\overline{\text{S1}}$, $\overline{\text{S0}}$ and COD/$\overline{\text{INTA}}$ are LOW and M/$\overline{\text{IO}}$ is HIGH. A_1 HIGH indicates halt, and A_1 LOW indicates shutdown. The 82288 bus controller does not issue ALE, nor is $\overline{\text{READY}}$ required to terminate a halt or shutdown bus operation.

During halt or shutdown, the 80286 may service PEREQ or HOLD requests. A processor extension segment overrun exception during shutdown will inhibit further service of PEREQ. Either NMI or RESET will force the 80286 out of either halt or shutdown. An INTR, if interrupts are enabled, or a processor extension segment overrun exception will also force the 80286 out of halt.



NOTES:

1. Data is ignored.
2. First INTA cycle should have at least one wait state inserted to meet 8259A minimum INTA pulse width.
3. Second INTA cycle must have at least one wait state inserted since the CPU will not drive $A_{23} - A_0$, $\overline{BHE}$, and $\overline{LOCK}$ until after the first TC state.

   The CPU imposed one/clock delay prevents bus contention between cascade address buffer being disabled by MCE ↓ and address outputs.

   Without the wait state, the 80286 address will not be valid for a memory cycle started immediately after the second INTA cycle. The 8259A also requires one wait state for minimum INTA pulse width.
4. $\overline{LOCK}$ is active for the first INTA cycle to prevent the 82289 from releasing the bus between INTA cycles in a multi-master system.
5. $A_{23} - A_0$ exits 3-state OFF during $\phi 2$ of the second T_C in the INTA cycle.

Figure 30. Interrupt Acknowledge Sequence

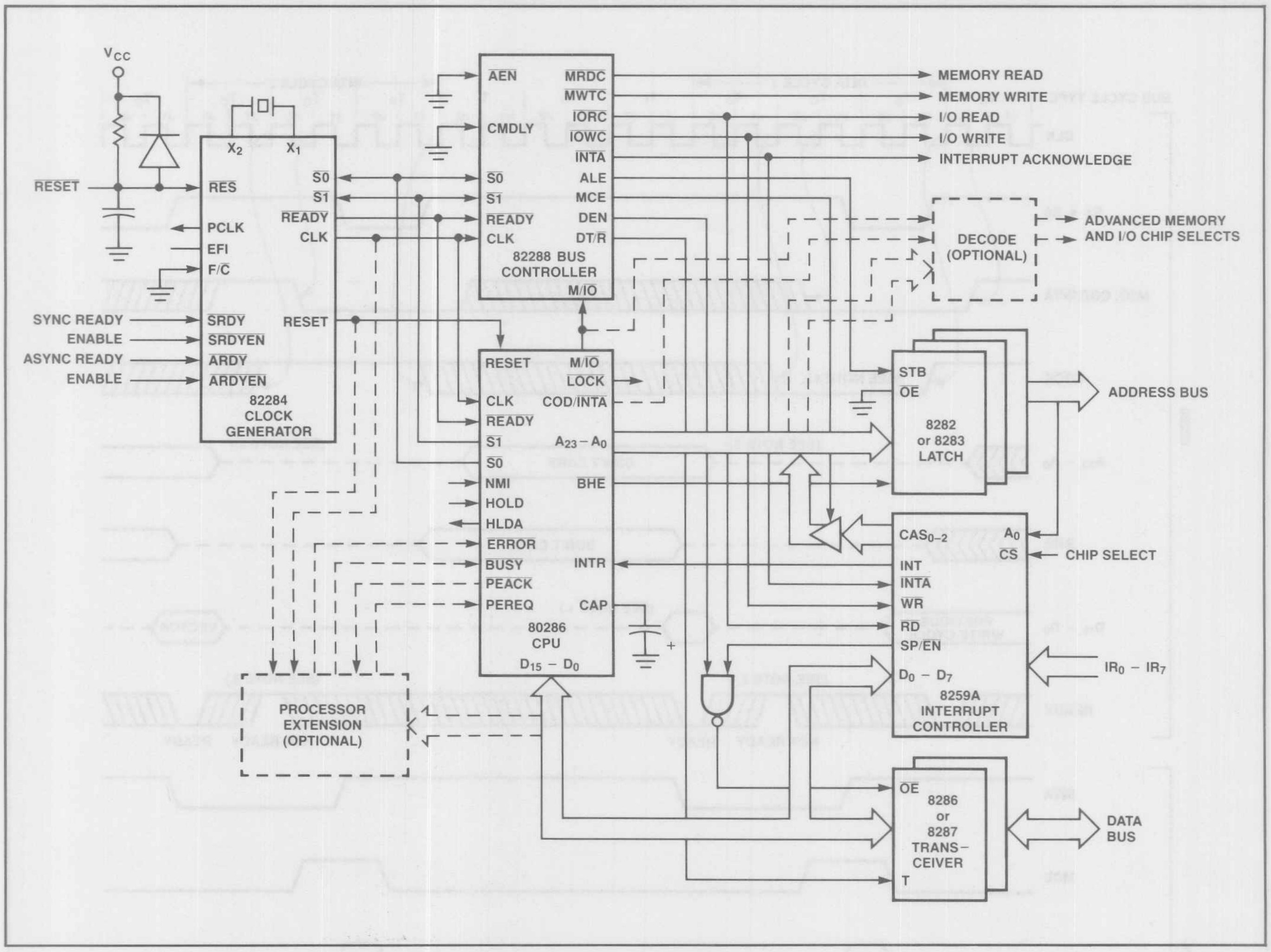


Figure 31. Basic iAPX 286 System Configuration

SYSTEM CONFIGURATIONS

The versatile bus structure of the iAPX 286 microsystem, with a full complement of support chips, allows flexible configuration of a wide range of systems. The basic configuration, shown in Figure 31, is similar to an iAPX 86 maximum mode system. It includes the CPU plus an 8259A interrupt controller, 82284 clock generator, and the 82288 Bus Controller. The iAPX 86 latches (8282 and 8283) and transceivers (8286 and 8287) may be used in an iAPX 286 microsystem.

As indicated by the dashed lines in Figure 31, the ability to add processor extensions is an integral feature of iAPX 286 microsystems. The processor extension interface allows external hardware to perform special functions and transfer data concurrent with CPU execution of other instructions. Full system integrity is maintained because the 80286 supervises all data transfers and instruction execution for the processor extension.

The iAPX 286/20 numeric data processor which includes the 80287 numeric processor extension (NPX) uses this interface. The iAPX 286/20 has all the instructions and data types of an iAPX 86/20 or iAPX 88/20. The 80287 NPX can perform numeric calculations and data transfers concurrently with CPU program execution. Numerics code and data have the same integrity as all other information protected by the iAPX 286 protection mechanism.

The 80286 can overlap chip select decoding and address propagation during the data transfer for the previous bus operation. This information is latched into the 8282/3's by ALE during the middle of a T_S cycle. The latched chip select and address information remains stable during the bus operation while the next cycles address is being decoded and propagated into the system. Decode logic can be implemented with a high speed bipolar PROM.

The optional decode logic shown in Figure 31 takes advantage of the overlap between address and data of the 80286 bus cycle to generate advanced memory and IO-select signals. This minimizes system performance

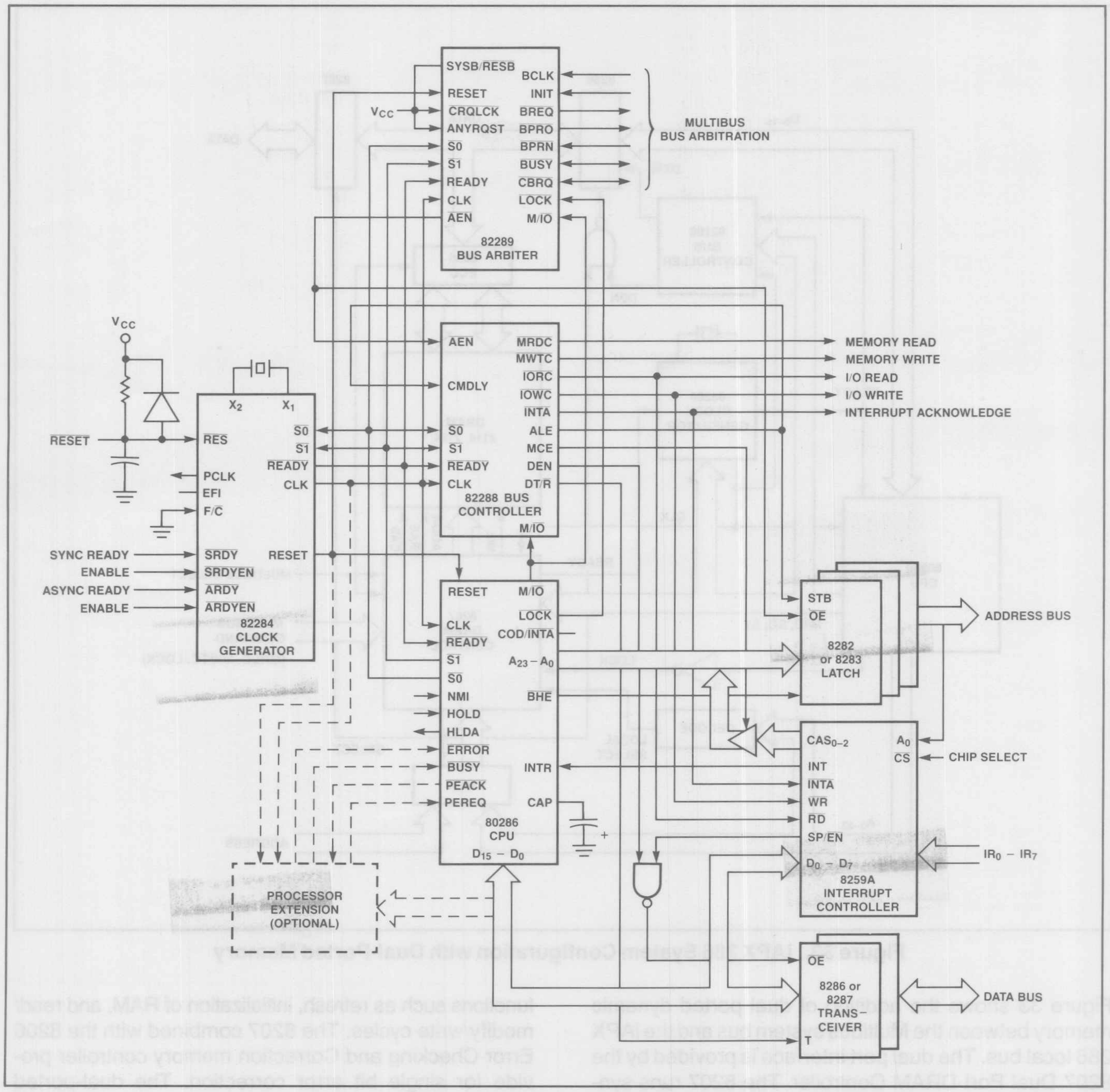


Figure 32. Multibus System Bus Interface

degradation caused by address propogation and decode delays. In addition to selecting memory and I/O, the advanced selects may be used with configurations supporting local and system buses to enable the appropriate bus interface for each bus cycle. The COD/$\overline{\text{INTA}}$ and M/$\overline{\text{IO}}$ signals are applied to the decode logic to distinguish between interrupt, I/O, code and data bus cycles.

By adding the 82289 bus arbiter chip the 80286 provides a Multibus system bus interface as shown in Figure 32. The ALE output of the 82288 for the Multibus bus is connected to its CMDLY input to delay the start of commands one system CLK as required to meet Multibus address and write data setup times. This arrangement will add at least one extra T_c state to each bus operation which uses the Multibus.

A second 82288 bus controller and additional latches and transceivers could be added to the local bus of Figure 32. This configuration allows the 80286 to support an on-board bus for local memory and peripherals, and the Multibus for system bus interfacing.

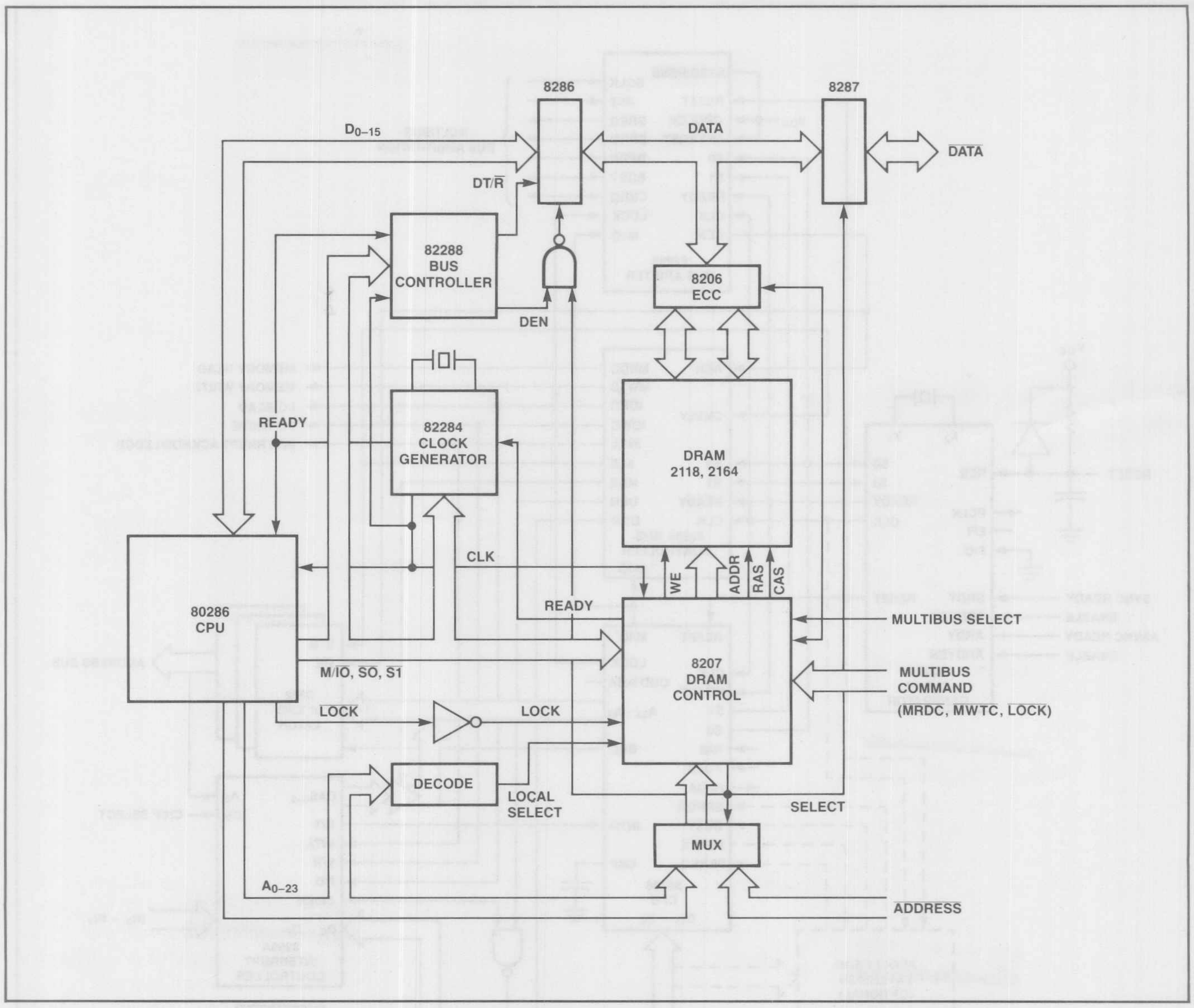


Figure 33. iAPX 286 System Configuration with Dual-Ported Memory

Figure 33 shows the addition of dual ported dynamic memory between the Multibus system bus and the iAPX 286 local bus. The dual port interface is provided by the 8207 Dual Port DRAM Controller. The 8207 runs synchronously with the CPU to maximize throughput for local memory references. It also arbitrates between requests from the local and system buses and performs functions such as refresh, initialization of RAM, and read/modify/write cycles. The 8207 combined with the 8206 Error Checking and Correction memory controller provide for single bit error correction. The dual-ported memory can be combined with a standard Multibus system bus interface to maximize performance and protection in multiprocessor system configurations.

PACKAGE

The 80286 is packaged in a 68-pin, leadless JEDEC type A hermetic chip carrier. Figure 34 illustrates the package, and Figure 2 shows the pinout.

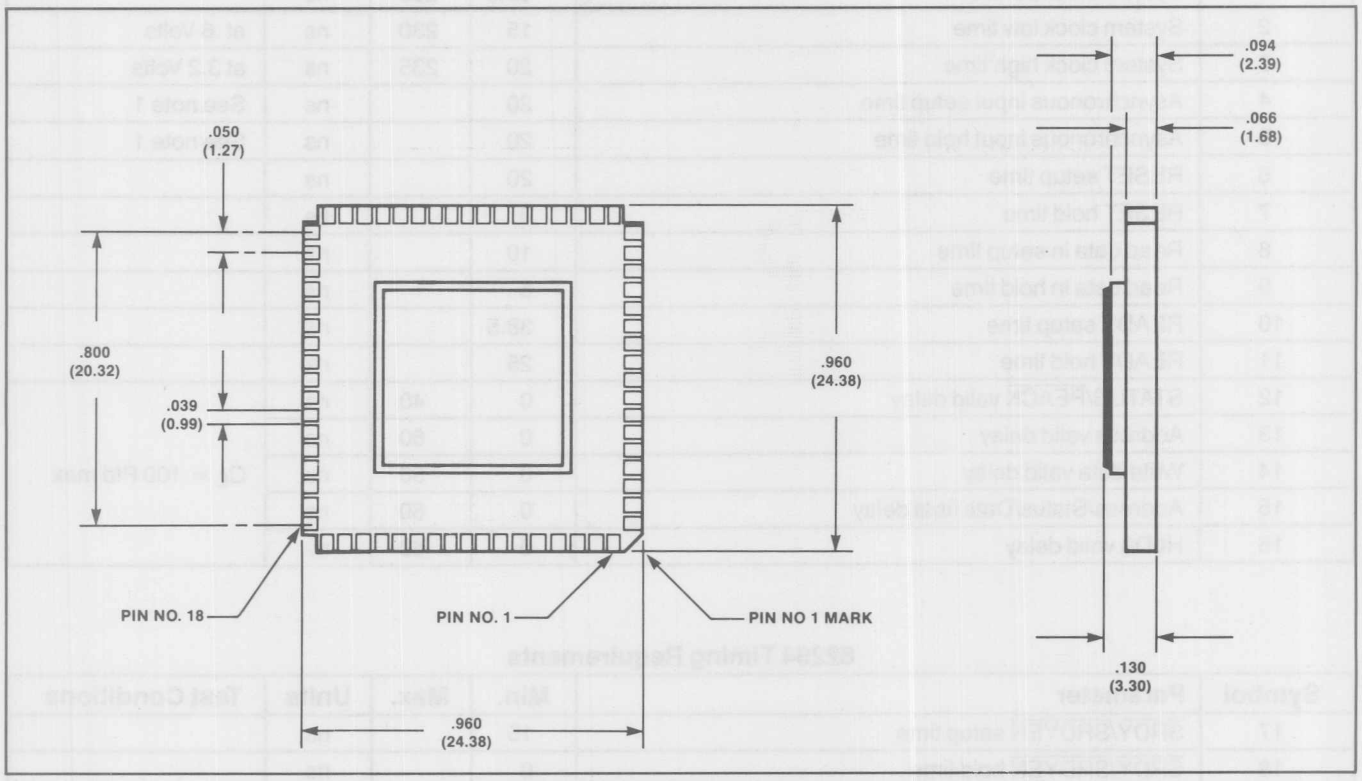


Figure 34. 80286 JEDEC Type A Package

ABSOLUTE MAXIMUM RATINGS*

Ambient Temperature Under Bias . . . . 0°C to 70°C
Storage Temperature . . . . −65°C to +150°C
Voltage on Any Pin with
  Respect to Ground . . . . −0.3 to +7V
Power Dissipation . . . . 3.6 Watt

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

D.C. CHARACTERISTICS (80286: $T_A = 0°C$ to $70°C$, $V_{CC} = 5V \pm 10\%$)

Symbol	Parameter	Min.	Max.	Units	Test Conditions
V_{IL}	Input Low Voltage	−0.5	+0.8	V	
V_{IH}	Input High Voltage	2.0	$V_{CC}+0.5$	V	
V_{OL}	Output Low Voltage		0.45	V	$I_{OL}=3.0$ mA
V_{OH}	Output High Voltage	2.4		V	$I_{OH}=-400$ μA
I_{CC}	Power Supply Current		600	mA	$T_A=25°C$
I_{LI}	Input Leakage Current		±10	μA	$0V \leq V_{IN} \leq V_{CC}$
I_{LO}	Output Leakage Current		±10	μA	$0.45V \leq V_{OUT} \leq V_{CC}$
V_{CL}	Clock Input Low voltage	−0.5	+0.6	V	
V_{CH}	Clock Input HIgh Voltage	3.8	$V_{CC}+1.0$	V	
C_{IN}	Capacitance of Inputs (All input except CLK)		10	pF	fc=1 MHz
C_O	Capacitance of I/O or outputs		20	pF	fc=1 MHz
C_{CLK}	Capacitance of CLK Input		12	pF	$f_c=1$ MHz

A.C. CHARACTERISTICS (T_A = 0°C to 70°C, V_{CC} = 5V ± 10%)

80286 Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
1	System clock period	62.5	250	ns	
2	System clock low time	15	230	ns	at .6 Volts
3	System clock high time	20	235	ns	at 3.2 Volts
4	Asynchronous input setup time	20		ns	See note 1
5	Asynchronous input hold time	20		ns	See note 1
6	RESET setup time	20		ns	
7	RESET hold time	0		ns	
8	Read data in setup time	10		ns	
9	Read data in hold time	5		ns	
10	READY setup time	38.5		ns	
11	READY hold time	25		nx	
12	STATUS/$\overline{\text{PEACK}}$ valid delay	0	40	ns	
13	Address valid delay	0	60	ns	
14	Write data valid delay	0	50	ns	C_L = 100 Pfd max
15	Address/Status/Data float delay	0	60	ns	
16	HLDA valid delay	0	60	ns	

82284 Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
17	$\overline{\text{SRDY}}/\overline{\text{SRDYEN}}$ setup time	15		ns	
18	$\overline{\text{SRDY}}/\overline{\text{SRDYEN}}$ hold time	0		ns	
19	$\overline{\text{ARDY}}/\overline{\text{ARDYEN}}$ setup time	−5		ns	See note 1
20	$\overline{\text{ARDY}}/\overline{\text{ARDYEN}}$ hold time	16		ns	See note 1.
21	PCLK delay	0	40	ns	C_L = 75 pfd I_{OL} = 5.25 ma I_{OH} = −1.05 ma

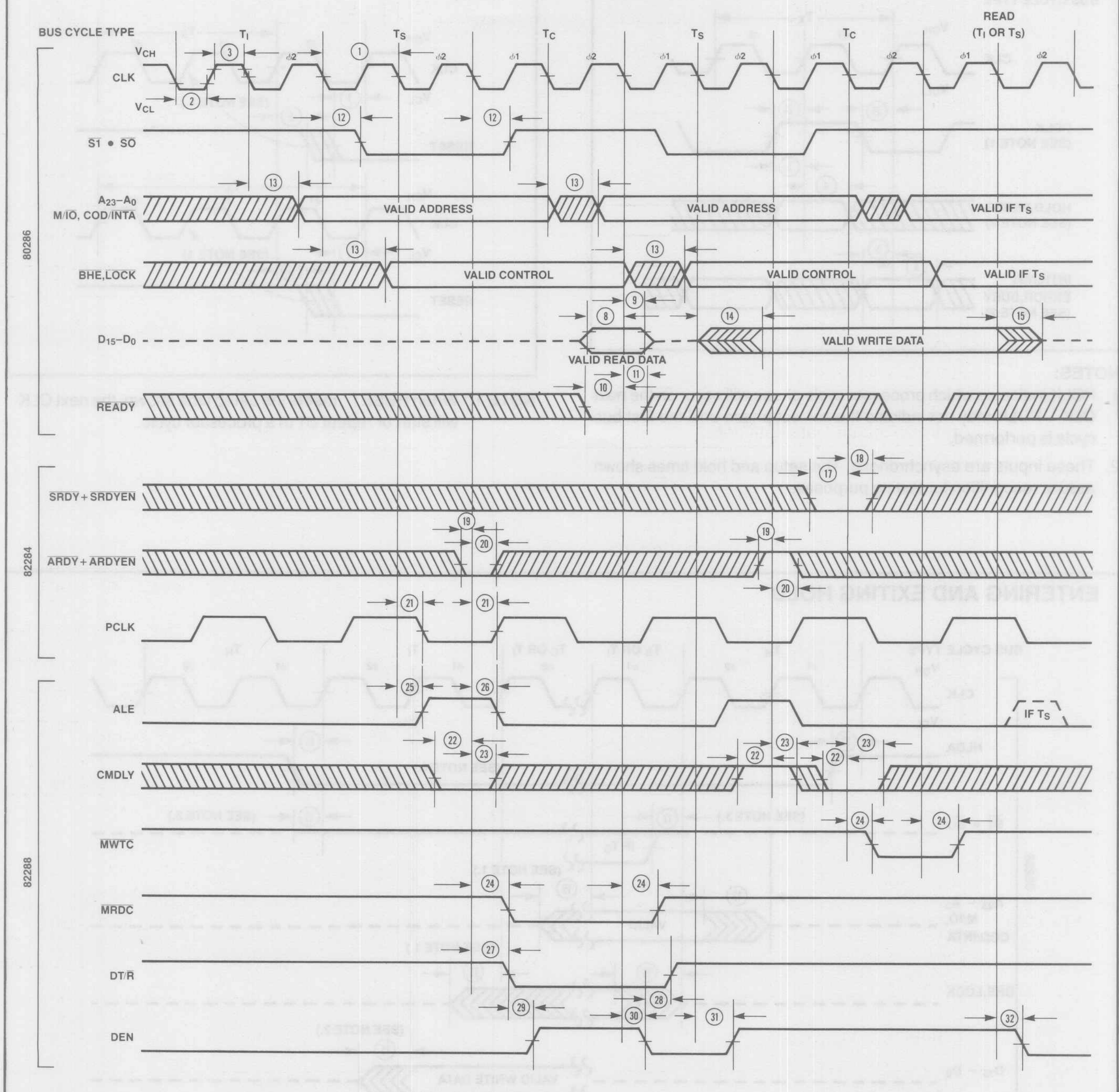
NOTE 1: These times are given for testing purposes to assure a predetermined action.

82288 Timing Requirements

Symbol	Parameter	Min.	Max.	Units	Test Conditions
22	CMDLY setup time	20		ns	
23	CMDLY hold time	0		ns	
24	Command delay	5	25	ns	C_L = 300 pfd max I_{OL} = 32 ma max I_{OH} = −5 ma max
25	ALE active delay	2.5	12	ns	
26	ALE inactive delay	0	15	ns	
27	DT/$\overline{\text{R}}$ read active delay	0	25	ns	
28	DT/$\overline{\text{R}}$ read inactive delay	15	40	ns	C_L = 80 pfd max I_{OL} = 16 ma max I_{OH} = −1 ma max
29	DEN read active delay	10	50	ns	
30	DEN read inactive delay	2.5	20	ns	
31	DEN write active delay	17.5	40	ns	
32	DEN write inactive delay	12.5	47.5	ns	

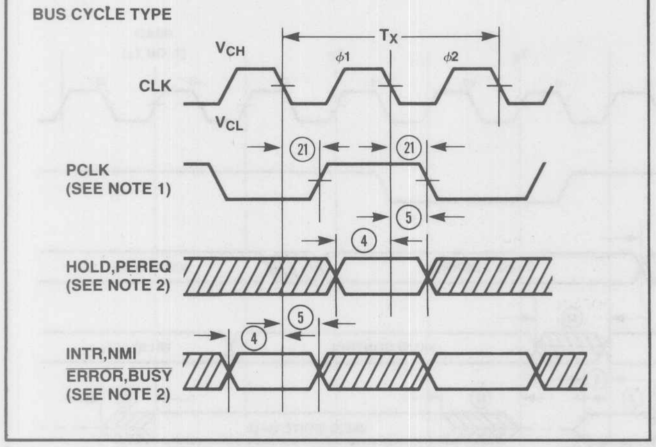
WAVEFORMS

MAJOR CYCLE TIMING



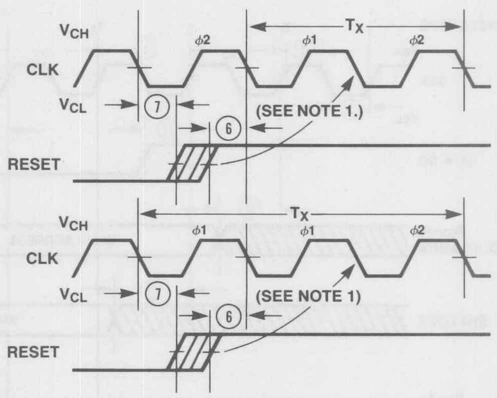
WAVEFORMS (Continued)

80286 ASYNCHRONOUS INPUT SIGNAL TIMING



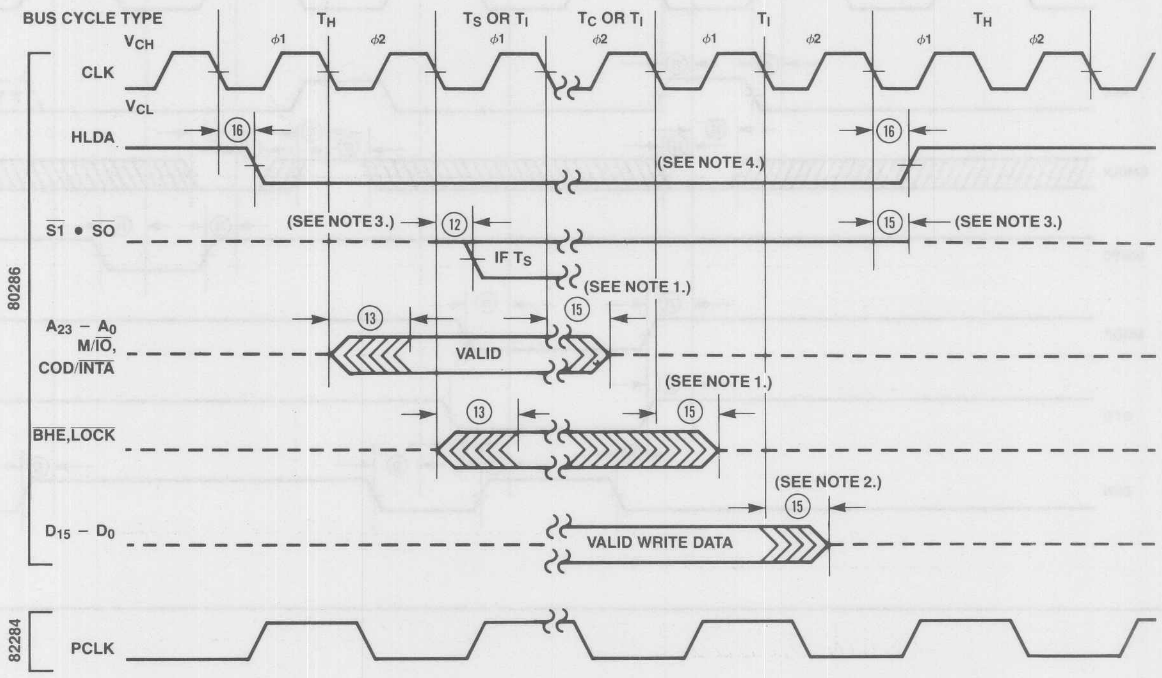
NOTES:
1. PCLK indicates which processor cycle phase will occur on the next CLK . PCLK may not indicate the correct phase until the first bus cycle is performed.
2. These inputs are asynchronous. The setup and hold times shown assure recognition for testing purposes.

80286 RESET INPUT TIMING AND SUBSEQUENT PROCESSOR CYCLE PHASE



NOTE 1: When RESET meets the setup time shown, the next CLK will start or repeat $\phi 1$ of a processor cycle.

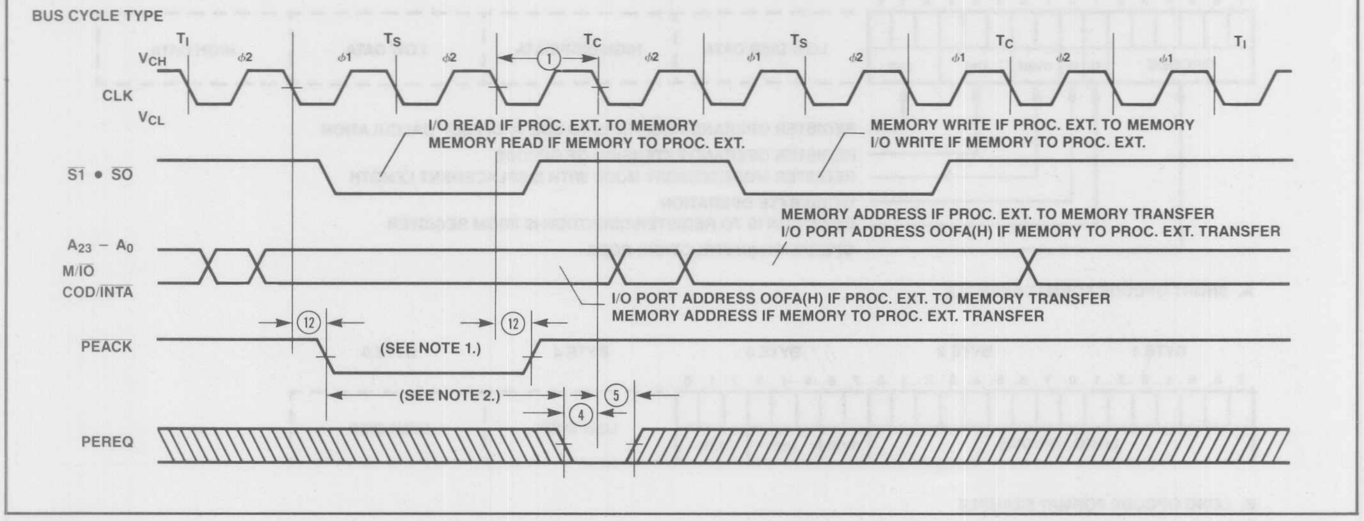
ENTERING AND EXITING HOLD



NOTES:
1. These signals may still be driven by the 80286 during the time shown. The worst case in terms of latest float time is shown.
2. The data bus will be driven as shown if the last cycle before T_I in the diagram was a write T_C.
3. The 80286 floats its status pins during T_H. External pullup resistors (in 82288) keep these signals high.
4. For HOLD request set up to HLDA, refer to Figure 29.

WAVEFORMS (Continued)

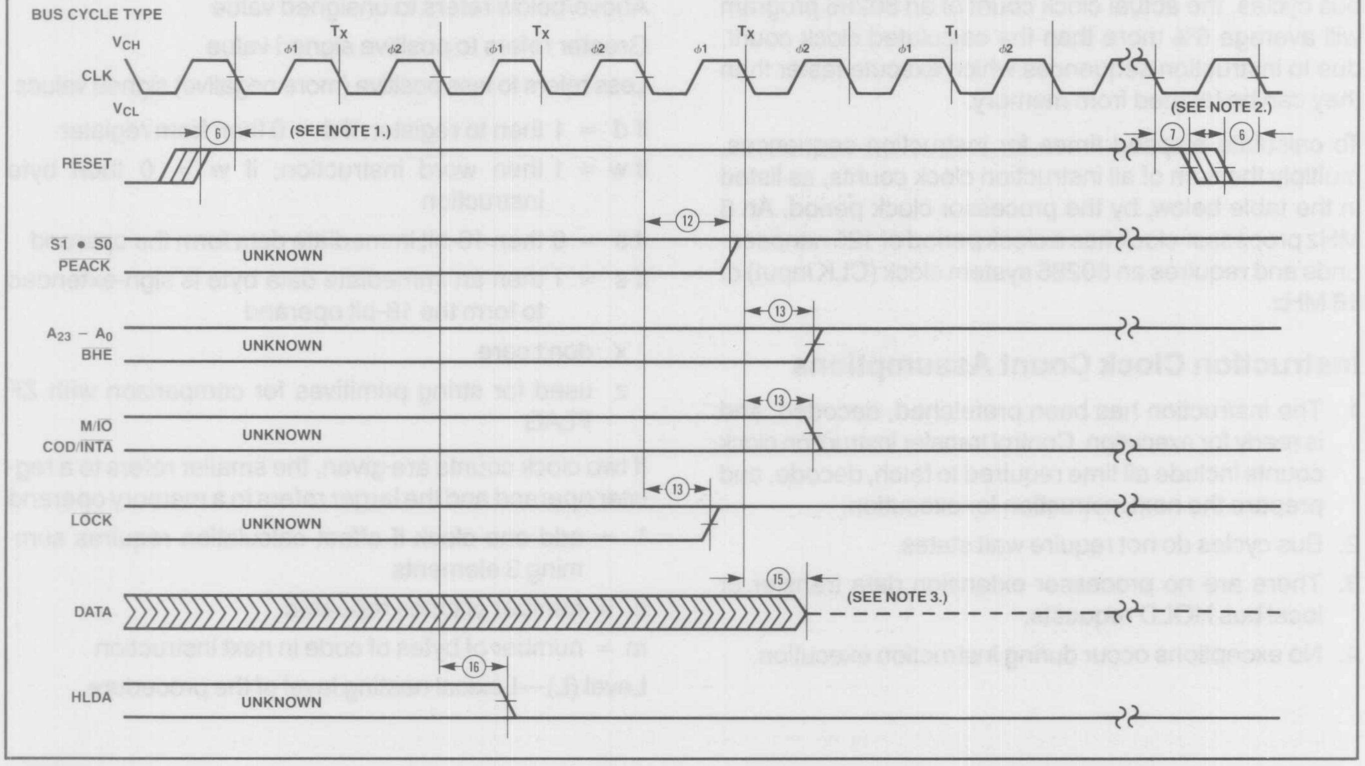
80286 PEREQ/PEACK TIMING REQUIRED PEREQ TIMING FOR ONE TRANSFER ONLY


BUS CYCLE TYPE
T_I
T_S
T_C
T_S
T_C
T_I
V_CH
CLK
V_CL
φ2
φ1
S1 • S0
I/O READ IF PROC. EXT. TO MEMORY
MEMORY READ IF MEMORY TO PROC. EXT.
MEMORY WRITE IF PROC. EXT. TO MEMORY
I/O WRITE IF MEMORY TO PROC. EXT.
A23 – A0
M/IO
COD/INTA
MEMORY ADDRESS IF PROC. EXT. TO MEMORY TRANSFER
I/O PORT ADDRESS OOFA(H) IF MEMORY TO PROC. EXT. TRANSFER
I/O PORT ADDRESS OOFA(H) IF PROC. EXT. TO MEMORY TRANSFER
MEMORY ADDRESS IF PROC. EXT. TO MEMORY TRANSFER
PEACK
(SEE NOTE 1.)
(SEE NOTE 2.)
PEREQ
1
4
5
12

NOTES:

1. PEACK always goes active during the first bus operation of a processor extension data operand transfer sequence. The first bus operation will be either a memory read at operand address or I/O read at port address OOFA(H).
2. To prevent a second processor extension data operand transfer, the worst case maximum time (Shown above) is: 3X① – ⑪ $_{max.}$ – ④ $_{min.}$ The actual, configuration dependent, maximum time is: 3X① – ⑪ $_{max.}$ – ④ $_{min.}$ + A X 2 X ①.
   A is the number of extra T_C states added to either the first or second bus operation of the processor extension data operand transfer sequence.

INITIAL 80286 PIN STATE DURING RESET



NOTES:

1. Setup time for RESET ↑ may be violated with the consideration that φ1 of the processor clock may begin one system CLK period later.
2. Setup and hold times for RESET ↓ must be met for proper operation.
3. The data bus is only guaranteed to be in 3-state OFF at the time shown.

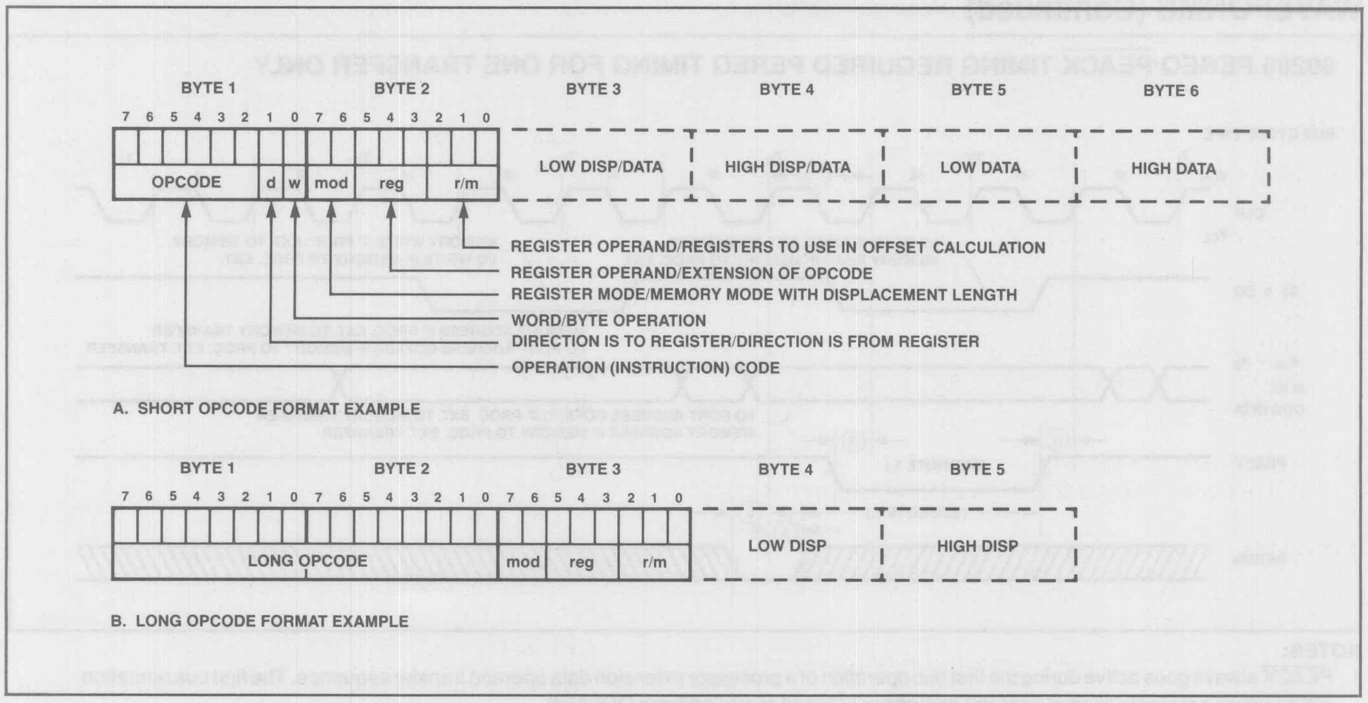

BYTE 1
BYTE 2
BYTE 3
BYTE 4
BYTE 5
BYTE 6
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0
OPCODE
d
w
mod
reg
r/m
LOW DISP/DATA
HIGH DISP/DATA
LOW DATA
HIGH DATA
REGISTER OPERAND/REGISTERS TO USE IN OFFSET CALCULATION
REGISTER OPERAND/EXTENSION OF OPCODE
REGISTER MODE/MEMORY MODE WITH DISPLACEMENT LENGTH
WORD/BYTE OPERATION
DIRECTION IS TO REGISTER/DIRECTION IS FROM REGISTER
OPERATION (INSTRUCTION) CODE
A. SHORT OPCODE FORMAT EXAMPLE
LONG OPCODE
LOW DISP
HIGH DISP
B. LONG OPCODE FORMAT EXAMPLE

Figure 35. 80286 Instruction Format Examples

80286 INSTRUCTION SET SUMMARY

Instruction Timing Notes

The instruction clock counts listed below establish the maximum execution rate of the 80286. With no delays in bus cycles, the actual clock count of an 80286 program will average 5% more than the calculated clock count, due to instruction sequences which execute faster than they can be fetched from memory.

To calculate elapsed times for instruction sequences, multiply the sum of all instruction clock counts, as listed in the table below, by the processor clock period. An 8 MHz processor clock has a clock period of 125 nanoseconds and requires an 80286 system clock (CLK input) of 16 MHz.

Instruction Clock Count Assumptions

1. The instruction has been prefetched, decoded, and is ready for execution. Control transfer instruction clock counts include all time required to fetch, decode, and prepare the next instruction for execution.
2. Bus cycles do not require wait states.
3. There are no processor extension data transfer or local bus HOLD requests.
4. No exceptions occur during instruction execution.

Instruction Set Summary Notes

Addressing displacements selected by the MOD field are not shown. If necessary they appear after the instruction fields shown.

Above/below refers to unsigned value

Greater refers to positive signed value

Less refers to less positive (more negative) signed values

if d = 1 then to register; if d = 0 then from register

if w = 1 then word instruction; if w = 0 then byte instruction

if s = 0 then 16-bit immediate data form the operand

if s = 1 then an immediate data byte is sign-extended to form the 16-bit operand

x don't care

z used for string primitives for comparison with ZF FLAG

If two clock counts are given, the smaller refers to a register operand and the larger refers to a memory operand

* = add one clock if offset calculation requires summing 3 elements

n = number of times repeated

m = number of bytes of code in next instruction

Level (L)—Lexical nesting level of the procedure

The following comments describe possible exceptions, side effects, and allowed usage for instructions in both operating modes of the 80286.

REAL ADDRESS MODE ONLY

1. This is a protected mode instruction. Attempted execution in real address mode will result in an undefined opcode exception (6).
2. A segment overrun exception (13) will occur if a word operand reference at offset FFFF(H) is attempted.
3. This instruction may be executed in real address mode to initialize the CPU for protected mode.
4. The IOPL and NT fields will remain 0.
5. Processor extension segment overrun interrupt (9) will occur if the operand exceeds the segment limit.

EITHER MODE

6. An exception may occur, depending on the value of the operand.
7. $\overline{\text{LOCK}}$ is automatically asserted regardless of the presence or absence of the LOCK instruction prefix.

PROTECTED VIRTUAL ADDRESS MODE ONLY

8. The destination of an INT, JMP, CALL, RET or IRET instruction must be in the defined limit of a code segment or a general protection exception (13) occurs.
9. A general protection exception (13) will occur if the memory operand can not be used due to either a segment limit or access rights violation. If a stack segment limit is violated, a stack segment overrun exception (12) occurs.
10. For segment load operations, the CPL, RPL, and DPL must agree with privilege rules to avoid an exception. The segment must be present to avoid a not-present exception (11). If the SS register is the destination, and a segment not-present violation occurs, a stack exception (12) occurs.
11. All segment descriptor accesses in the GDT or LDT made by this instruction will automatically assert $\overline{\text{LOCK}}$ to maintain descriptor integrity in multiprocessor systems.
12. JMP, CALL, INT, RET, IRET instructions referring to another code segment will cause a general protection exception (13) if any privilege rule is violated.
13. A general protection exception (13) occurs if $\text{CPL} \neq 0$.
14. A general protection exception (13) occurs if $\text{CPL} > \text{IOPL}$.
15. The IF field of the flag word is not updated if $\text{CPL} > \text{IOPL}$. The IOPL field is updated only if $\text{CPL} = 0$.
16. Any violation of privilege rules as applied to the selector operand do not cause a protection exception; rather, the instruction does not return a result and the zero flag is cleared.
17. If the starting address of the memory operand violates a segment limit, or an invalid access is attempted, a general protection exception (13) will occur before the ESC instruction is executed. A stack segment overrun exception (12) will occur if the stack limit is violated by the operand's starting address. If a segment limit is violated during an attempted data transfer then a processor extension segment overrun exception (9) occurs.

80286 INSTRUCTION SET SUMMARY

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
DATA TRANSFER								
MOV = Move:								
Register to Register/Memory	1 0 0 0 1 0 0 w	mod reg r/m			2,3*	2,3*	2	9
Register/memory to register	1 0 0 0 1 0 1 w	mod reg r/m			2,5*	2,5*	2	9
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	2,3*	2,3*	2	9
Immediate to register	1 0 1 1 w reg	data	data if w = 1		2	2		
Memory to accumulator	1 0 1 0 0 0 0 w	addr-low	addr-high		5	5	2	9
Accumulator to memory	1 0 1 0 0 0 1 w	addr-low	addr-high		3	3	2	9
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 reg r/m			2,5*	17,19*	2	9,10,11
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 reg r/m			2,3*	2,3*	2	9
PUSH = Push:								
Memory	1 1 1 1 1 1 1 1	mod 1 1 0 r/m			5*	5*	2	9
Register	0 1 0 1 0 reg				3	3	2	9
Segment register	0 0 0 reg 1 1 0				3	3	2	9
Immediate	0 1 1 0 1 0 s 0	data	data if s = 0		3	3	2	9
PUSHA = Push All	0 1 1 0 0 0 0 0				17	17	2	9
POP = Pop:								
Memory	1 0 0 0 1 1 1 1	mod 0 0 0 r/m			5*	5*	2	9
Register	0 1 0 1 1 reg				5	5	2	9
Segment register	0 0 0 reg 1 1 1	(reg ≠ 01)			5	20	2	9,10,11
POPA = Pop All	0 1 1 0 0 0 0 1				19	19	2	9
XCHG = Exchange:								
Register/memory with register	1 0 0 0 0 1 1 w	mod reg r/m			3,5*	3,5*	2,7	7,9
Register with accumulator	1 0 0 1 0 reg				3	3		
IN = Input from:								
Fixed port	1 1 1 0 0 1 0 w	port			5	5		14
Variable port	1 1 1 0 1 1 0 w				5	5		14
OUT = Output to:								
Fixed port	1 1 1 0 0 1 1 w	port			3	3		14
Variable port	1 1 1 0 1 1 1 w				3	3		14
XLAT = Translate byte to AL	1 1 0 1 0 1 1 1				5	5		9
LEA = Load EA to register	1 0 0 0 1 1 0 1	mod reg r/m			3*	3*		
LDS = Load pointer to DS	1 1 0 0 0 1 0 1	mod reg r/m	(mod ≠ 11)		7*	21*	2	9,10,11
LES = Load pointer to ES	1 1 0 0 0 1 0 0	mod reg r/m	(mod ≠ 11)		7*	21*	2	9,10,11
LAHF = Load AH with flags	1 0 0 1 1 1 1 1				2	2		
SAHF = Store AH into flags	1 0 0 1 1 1 1 0				2	2		
PUSHF = Push flags	1 0 0 1 1 1 0 0				3	3	2	9
POPF = Pop flags	1 0 0 1 1 1 0 1				5	5	2,4	9,15

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC								
ADD = Add:								
Reg/memory with register to either	0 0 0 0 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 0 0 r/m	data	data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 0 1 0 w	data	data if w = 1		3	3		
ADC = Add with carry:								
Reg/memory with register to either	0 0 0 1 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 s w	mod 0 1 0 r/m	data	data if s w = 01	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 1 0 1 0 w	data	data if w = 1		3	3		
INC = Increment:								
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 0 r/m			2,7*	2,7*	2	9
Register	0 1 0 0 0 reg				2	2		
SUB = Subtract:								
Reg/memory and register to either	0 0 1 0 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w	mod 1 0 1 r/m	data	data if s w = 0 1	3,7*	3,7*	2	9
Immediate from accumulator	0 0 1 0 1 1 0 w	data	data if w = 1		3	3		
SBB = Subtract with borrow:								
Reg/memory and register to either	0 0 0 1 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate from register/memory	1 0 0 0 0 0 s w	mod 0 1 1 r/m	data	data if s w = 0 1	3,7*	3,7*	2	9
Immediate from accumulator	0 0 0 1 1 1 0 w	data	data if w = 1		3	3		
DEC = Decrement:								
Register/memory	1 1 1 1 1 1 1 w	mod 0 0 1 r/m			2,7*	2,7*	2	9
Register	0 1 0 0 1 reg				2	2		
CMP = Compare:								
Register/memory with register	0 0 1 1 1 0 1 w	mod reg r/m			2,6*	2,6*	2	9
Register with register/memory	0 0 1 1 1 0 0 w	mod reg r/m			2,7*	2,7*	2	9
Immediate with register/memory	1 0 0 0 0 0 s w	mod 1 1 1 r/m	data	data if s w = 0 1	3,6*	3,6*	2	9
Immediate with accumulator	0 0 1 1 1 1 0 w	data	data if w = 1		3	3		
NEG = Change sign	1 1 1 1 0 1 1 w	mod 0 1 1 r/m			2	7*	2	9
AAA = ASCII adjust for add	0 0 1 1 0 1 1 1				3	3		
DAA = Decimal adjust for add	0 0 1 0 0 1 1 1				3	3		
AAS = ASCII adjust for subtract	0 0 1 1 1 1 1 1				3	3		
DAS = Decimal adjust for subtract	0 0 1 0 1 1 1 1				3	3		
MUL = Multiply (unsigned):	1 1 1 1 0 1 1 w	mod 1 0 0 r/m						
Register-Byte					13	13		
Register-Word					21	21		
Memory-Byte					16*	16*	2	9
Memory-Word					24*	24*	2	9
IMUL = Integer multiply (signed):	1 1 1 1 0 1 1 w	mod 1 0 1 r/m						
Register-Byte					13	13		
Register-Word					21	21		
Memory-Byte					16*	16*	2	9
Memory-Word					24*	24*	2	9
IMUL = Integer immediate multiply (signed)	0 1 1 0 1 0 s 1	mod reg r/m	data	data if s = 0	21,24*	21,24*	2	9
DIV = Divide (unsigned):	1 1 1 1 0 1 1 w	mod 1 1 0 r/m						
Register-Byte					14	14		
Register-Word					22	22		
Memory-Byte					17*	17*	2,6	6,9
Memory-Word					25*	25*	2,6	6,9

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
ARITHMETIC (Continued):								
IDIV = Integer divide (signed):	1 1 1 1 0 1 1 w	mod 1 1 1 r/m						
Register-Byte					17	17		
Register-Word					25	25		
Memory-Byte					20*	20*	2	9
Memory-Word					28*	28*	2	9
AAM = ASCII adjust for multiply	1 1 0 1 0 1 0 0	0 0 0 0 1 0 1 0			16	16		
AAD = ASCII adjust for divide	1 1 0 1 0 1 0 1	0 0 0 0 1 0 1 0			14	14		
CBW = Convert byte to word	1 0 0 1 1 0 0 0				2	2		
CWD = Convert word to double word	1 0 0 1 1 0 0 1				2	2		
LOGIC								
Shift/Rotate Instructions:								
Register/Memory by 1	1 1 0 1 0 0 0 w	mod TTT r/m			2,7*	2,7*	2	9
Register/Memory by CL	1 1 0 1 0 0 1 w	mod TTT r/m			5+n,8+n*	5+n,8+n*	2	9
Register/Memory by Count	1 1 0 0 0 0 0 w	mod TTT r/m	count		5+n,8+n*	5+n,8+n*	2	9
		TTT	**Instruction**					
		0 0 0	ROL					
		0 0 1	ROR					
		0 1 0	RCL					
		0 1 1	RCR					
		1 0 0	SHL/SAL					
		1 0 1	SHR					
		1 1 1	SAR					
AND = And:								
Reg/memory and register to either	0 0 1 0 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 0 0 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 0 0 1 0 w	data	data if w = 1		3	3		
TEST = And function to flags, no result:								
Register/memory and register	1 0 0 0 0 1 0 w	mod reg r/m			2,6*	2,6*	2	9
Immediate data and register/memory	1 1 1 1 0 1 1 w	mod 0 0 0 r/m	data	data if w = 1	3,6*	3,6*	2	9
Immediate data and accumulator	1 0 1 0 1 0 0 w	data	data if w = 1		3	3		
OR = Or:								
Reg/memory and register to either	0 0 0 0 1 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 0 0 1 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 0 0 1 1 0 w	data	data if w = 1		3	3		
XOR = Exclusive or:								
Reg/memory and register to either	0 0 1 1 0 0 d w	mod reg r/m			2,7*	2,7*	2	9
Immediate to register/memory	1 0 0 0 0 0 0 w	mod 1 1 0 r/m	data	data if w = 1	3,7*	3,7*	2	9
Immediate to accumulator	0 0 1 1 0 1 0 w	data	data if w = 1		3	3		
NOT = Invert register/memory	1 1 1 1 0 1 1 w	mod 0 1 0 r/m			2,7*	2,7*	2	9
STRING MANIPULATION:								
MOVS = Move byte/word	1 0 1 0 0 1 0 w				5	5	2	9
CMPS = Compare byte/word	1 0 1 0 0 1 1 w				8	8	2	9
SCAS = Scan byte/word	1 0 1 0 1 1 1 w				7	7	2	9
LODS = Load byte/wd to AL/AX	1 0 1 0 1 1 0 w				5	5	2	9
STOS = Stor byte/wd from AL/A	1 0 1 0 1 0 1 w				3	3	2	9
INS = Input byte/wd from DX port	0 1 1 0 1 1 0 w				5	5	2	9,14
OUTS = Output byte/wd to DX port	0 1 1 0 1 1 1 w				5	5	2	9,14

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
STRING MANIPULATION (Continued):								
Repeated by count in CX								
MOVS = Move string	1 1 1 1 0 0 1 0	1 0 1 0 0 1 0 w			5+4n	5+4n	2	9
CMPS = Compare string	1 1 1 1 0 0 1 z	1 0 1 0 0 1 1 w			5+9n	5+9n	2	9
SCAS = Scan string	1 1 1 1 0 0 1 z	1 0 1 0 1 1 1 w			5+8n	5+8n	2	9
LODS = Load string	1 1 1 1 0 0 1 0	1 0 1 0 1 1 0 w			5+4n	5+4n	2	9
STOS = Store string	1 1 1 1 0 0 1 0	1 0 1 0 1 0 1 w			4+3n	4+3n	2	9
INS = Input string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 0 w			5+4n	5+4n	2	9,14
OUTS = Output string	1 1 1 1 0 0 1 0	0 1 1 0 1 1 1 w			5+4n	5+4n	2	9,14
CONTROL TRANSFER								
CALL = Call:								
Direct within segment	1 1 1 0 1 0 0 0	disp-low	disp-high		7+m	7+m	2	8
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 0 1 0 r/m			7+m,11+m*	7+m,11+m*	2	8,9
Direct intersegment	1 0 0 1 1 0 1 0	segment offset			13+m	26+m	2	8,11,12
		segment selector						
Protected Mode Only (Direct intersegment):								
Via call gate to same privilege level						41+m		8,11,12
Via call gate to different privilege level, no parameters						82+m		8,11,12
Via call gate to different privilege level, x parameters						86+4x+m		8,11,12
Via TSS						177+m		8,11,12
Via task gate						182+m		8,11,12
Indirect intersegment	1 1 1 1 1 1 1 1	mod 0 1 1 r/m		(mod ≠ 11)	16+m	29+m*	2	8,9,11,12
Protected Mode Only (Indirect intersegment):								
Via call gate to same privilege level						44+m*		8,9,11,12
Via call gate to different privilege level, no parameters						83+m*		8,9,11,12
Via call gate to different privilege level, x parameters						90+4x+m*		8,9,11,12
Via TSS						180+m*		8,9,11,12
Via task gate						185+m*		8,9,11,12
JMP = Unconditional jump:								
Short/long	1 1 1 0 1 0 1 1	disp-low			7+m	7+m		8
Direct within segment	1 1 1 0 1 0 0 1	disp-low	disp-high		7+m	7+m		8
Register/memory indirect within segment	1 1 1 1 1 1 1 1	mod 1 0 0 r/m			7+m,11+m*	7+m,11+m*	2	8,9
Direct intersegment	1 1 1 0 1 0 1 0	segment offset			11+m	23+m		8,11,12
		segment selector						
Protected Mode Only (Direct intersegment):								
Via call gate to same privilege level						38+m		8,11,12
Via TSS						175+m		8,11,12
Via task gate						180+m		8,11,12
Indirect intersegment	1 1 1 1 1 1 1 1	mod 1 0 1 r/m		(mod ≠ 11)	15+m*	26+m*	2	8,9,11,12
Protected Mode Only (Indirect intersegment):								
Via call gate to same privilege level						41+m*		8,9,11,12
Via TSS						178+m*		8,9,11,12
Via task gate						183+m*		8,9,11,12
RET = Return from CALL:								
Within segment	1 1 0 0 0 0 1 1				11+m	11+m	2	8,9
Within seg adding immed to SP	1 1 0 0 0 0 1 0	data-low	data-high		11+m	11+m	2	8,9
Intersegment	1 1 0 0 1 0 1 1				15+m	25+m	2	8,9,11,12
Intersegment adding immediate to SP	1 1 0 0 1 0 1 0	data-low	data-high		15+m		2	8,9,11,12
Protected Mode Only (RET):								
To different privilege level						55+m		

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT				CLOCK COUNT		COMMENTS	
					Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
CONTROL TRANSFER (Continued):								
JE/JZ = Jump on equal/zero	0 1 1 1 0 1 0 0	disp			7+m or 3	7+m or 3		8
JL/JNGE = Jump on less/not greater or equal	0 1 1 1 1 1 0 0	disp			7+m or 3	7+m or 3		8
JLE/JNG = Jump on less or equal/not greater	0 1 1 1 1 1 1 0	disp			7+m or 3	7+m or 3		8
JB/JNAE = Jump on below/not above or equal	0 1 1 1 0 0 1 0	disp			7+m or 3	7+m or 3		8
JBE/JNA = Jump on below or equal/not above	0 1 1 1 0 1 1 0	disp			7+m or 3	7+m or 3		8
JP/JPE = Jump on parity/parity even	0 1 1 1 1 0 1 0	disp			7+m or 3	7+m or 3		8
JO = Jump on overflow	0 1 1 1 0 0 0 0	disp			7+m or 3	7+m or 3		8
JS = Jump on sign	0 1 1 1 1 0 0 0	disp			7+m or 3	7+m or 3		8
JNE/JNZ = Jump on not equal/not zero	0 1 1 1 0 1 0 1	disp			7+m or 3	7+m or 3		8
JNL/JGE = Jump on not less/greater or equal	0 1 1 1 1 1 0 1	disp			7+m or 3	7+m or 3		8
JNLE/JG = Jump on not less or equal/greater	0 1 1 1 1 1 1 1	disp			7+m or 3	7+m or 3		8
JNB/JAE = Jump on not below/above or equal	0 1 1 1 0 0 1 1	disp			7+m or 3	7+m or 3		8
JNBE/JA = Jump on not below or equal/above	0 1 1 1 0 1 1 1	disp			7+m or 3	7+m or 3		8
JNP/JPO = Jump on not par/par odd	0 1 1 1 1 0 1 1	disp			7+m or 3	7+m or 3		8
JNO = Jump on not overflow	0 1 1 1 0 0 0 1	disp			7+m or 3	7+m or 3		8
JNS = Jump on not sign	0 1 1 1 1 0 0 1	disp			7+m or 3	7+m or 3		8
LOOP = Loop CX times	1 1 1 0 0 0 1 0	disp			8+m or 4	8+m or 4		8
LOOPZ/LOOPE = Loop while zero/equal	1 1 1 0 0 0 0 1	disp			8+m or 4	8+m or 4		8
LOOPNZ/LOOPNE = Loop while not zero/equal	1 1 1 0 0 0 0 0	disp			8+m or 4	8+m or 4		8
JCXZ = Jump on CX zero	1 1 1 0 0 0 1 1	disp			8+m or 4	8+m or 4		8
ENTER = Enter Procedure	1 1 0 0 1 0 0 0	data-low	data-high	L				
L = 0					11	11	2	9
L = 1					15	15	2	9
L > 1					16+4(L−1)	16+4(L−1)	2	9
LEAVE = Leave Procedure	1 1 0 0 1 0 0 1				5	5	2	9
INT = Interrupt:								
Type specified	1 1 0 0 1 1 0 1	type			23+m		2	
Type 3	1 1 0 0 1 1 0 0				23+m		2	
INTO = Interrupt on overflow	1 1 0 0 1 1 1 0				24+m or 3 (3 if no interrupt)	24+m or 3 (3 if no interrupt)	2	
Protected Mode Only:								
Via interrupt or trap gate to same privilege level						40+m		8,11,12
Via interrupt or trap gate to fit different privilege level						78+m		8,11,12
Via Task Gate						167+m		8,11,12
IRET = Interrupt return	1 1 0 0 1 1 1 1				17+m	31+m	2,4	8,9,11,12,15
Protected Mode Only:								
To different privilege level						55+m		8,9,11,12,15
To different task (NT = 1)						169+m		8,9,11,12
BOUND = Detect value out of range	0 1 1 0 0 0 1 0	mod reg r/m			13*	13* (Use INT clock count if exception 5)	2,6	6,8,9,11,12

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

80286 INSTRUCTION SET SUMMARY (Continued)

FUNCTION	FORMAT			CLOCK COUNT		COMMENTS	
				Real Address Mode	Protected Virtual Address Mode	Real Address Mode	Protected Virtual Address Mode
PROCESSOR CONTROL							
CLC = Clear carry	1 1 1 1 1 0 0 0			2	2		
CMC = Complement carry	1 1 1 1 0 1 0 1			2	2		
STC = Set carry	1 1 1 1 1 0 0 1			2	2		
CLD = Clear direction	1 1 1 1 1 1 0 0			2	2		
STD = Set direction	1 1 1 1 1 1 0 1			2	2		
CLI = Clear interrupt	1 1 1 1 1 0 1 0			3	3		14
STI = Set interrupt	1 1 1 1 1 0 1 1			2	2		14
HLT = Halt	1 1 1 1 0 1 0 0			2	2		13
WAIT = Wait	1 0 0 1 1 0 1 1			3	3		
LOCK = Bus lock prefix	1 1 1 1 0 0 0 0			0	0		14
CTS = Clear task switched flag	0 0 0 0 1 1 1 1	0 0 0 0 0 1 1 0		2	2	3	13
ESC = Processor Extension Escape	1 0 0 1 1 T T T	mod LLL r/m		9-20*	9-20*	5	17
	(TTT LLL are opcode to processor extension)						
PROTECTION CONTROL							
LGDT = Load global descriptor table register	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 0 r/m	11*	11*	2,3	9,13
SGDT = Store global descriptor table register	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 0 r/m	11*	11*	2,3	9
LIDT = Load interrupt descriptor table register	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 1 1 r/m	12*	12*	2,3	9,13
SIDT = Store interrupt descriptor table register	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 0 0 1 r/m	12*	12*	2,3	9
LLDT = Load local descriptor table register from register memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 010 r/m		17,19*	1	9,11,13
SLDT = Store local descriptor table register to register/memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 0 r/m		2,3*	1	9
LTR = Load task register from register/memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 1 1 r/m		17,19*	1	9,11,13
STR = Store task register to register memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 0 0 1 r/m		2,3*	1	9,11,13
LMSW = Load machine status word from register/memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 1 0 r/m	3,6*	3,6*	2,3	9,13
SMSW = Store machine status word	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 1	mod 1 0 0 r/m	2,3*	2,3*	2,3	9
LAR = Load access rights from register/memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 0	mod reg r/m		14,16*	1	9,16
LSL = Load segment limit from register/memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 1 1	mod reg r/m		14,16*	1	9,16
ARPL = Adjust requested privilege level: from register/memory		0 1 1 0 0 0 1 1	mod reg r/m		10*,11*	2	9
VERR = Verify read access: register/memory	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 1 0 0 r/m		14,16*	1	9,16
VERR = Verify write access:	0 0 0 0 1 1 1 1	0 0 0 0 0 0 0 0	mod 1 0 1 r/m		14,16*	1	9,16

Shaded areas indicate instructions not available in iAPX 86, 88 microsystems.

Footnotes

The effective Address (EA) of the memory operand is computed according to the mod and r/m fields:

if mod = 11 then r/m is treated as a REG field
if mod = 00 then DISP = 0*, disp-low and disp-high are absent
if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
if mod = 10 then DISP = disp-high: disp-low

if r/m = 000 then EA = (BX) + (SI) + DISP
if r/m = 001 then EA = (BX) + (DI) + DISP
if r/m = 010 then EA = (BP) + (SI) + DISP
if r/m = 011 then EA = (BP) + (DI) + DISP
if r/m = 100 then EA = (SI) + DISP
if r/m = 101 then EA = (DI) + DISP
if r/m = 110 then EA = (BP) + DISP*
if r/m = 111 then EA = (BX) + DISP

DISP follows 2nd byte of instruction (before data if required)

*except if mod = 00 and r/m = 110 then EA = disp-high: disp-low.

SEGMENT OVERRIDE PREFIX

$\boxed{0\ 0\ 1\ \text{reg}\ 1\ 1\ 0}$

reg is assigned according to the following:

reg	Segment Register
00	ES
01	CS
10	SS
11	DS

REG is assigned according to the following table:

16-Bit (w = 1)		8-Bit (w = 0)	
000	AX	000	AL
001	CX	001	CL
010	DX	010	DL
011	BX	011	BL
100	SP	100	AH
101	BP	101	CH
110	SI	110	DH
111	DI	111	BH

The physical addresses of all operands addressed by the BP register are computed using the SS segment register. The physical addresses of the destination operands of the string primitive operations (those addressed by the DI register) are computed using the ES segment, which may not be overridden.